Smart Stores Technology

Tecnología para Tiendas Inteligentes

By
Arturo Acuaviva Huertos
Inmaculada Pérez Garbín

Double Degree in Computer Science and Mathematics
Faculty of Computer Science

Supervised by
Belén Díaz Agudo
Antonio A. Sánchez Ruiz-Granados

Madrid, 2020–2021

Smart Stores Technology

Tecnología para Tiendas Inteligentes

Computer Science's Final Project

Arturo Acuaviva Huertos & Inmaculada Pérez Garbín

Supervised by

Belén Díaz Agudo

Antonio A. Sánchez Ruiz-Granados

Department of Software Engineering and Artificial Intelligence
Faculty of Computer Science
Complutense University of Madrid

Madrid, 2021

Acknowledgements

We thank our supervisors, Belén Díaz Agudo and Antonio A. Sánchez Ruiz-Granados, from the Department of Software Engineering and Artificial Intelligence of Complutense University of Madrid, who provided insight and expertise that greatly assisted this research. We are grateful for the trust placed in us to carry out a project, a priori, unknown and ambitious, for which we have been given free hand to experiment and learn. We will never forget those Friday meetings in which time flew talking not only about the project, but also about other interesting fields and topics.

We also thank José Ignacio Herguedas Fernández for his assistance during the design and handcrafting of the hardware used in this project.

Finally, we wish to extend our special thanks to our families. They have been our travelling companion through our whole degrees and lives, supporting and advising us every day. Thanks for providing us unending inspiration, wise counsels and sympathetic ear.

Abstract

Smart stores technologies exemplify how Artificial Intelligence and Internet of Things can effectively join forces to shape the future of retailing. With an increasing number of companies proposing and implementing their own smart store concepts, such as Amazon Go or Tao Cafe, a new field is clearly emerging. Since the technologies used to build their infrastructure offer significant competitive advantages, companies are not publicly sharing their own designs. For this reason, this work presents a new smart store model named Mercury, which aims to take the edge off of the lack of public and accessible information and research documents in this field. We do not only introduce a comprehensive smart store model, but also work-through a feasible detailed implementation so that anyone can build their own system upon it.

Keywords— smart stores, cashier-less, cashier-free, internet of things, artificial intelligence, smart retailing, unstaffed retail, auto-checkout, sensor fusion, just walk out

Resumen

Las tecnologías utilizadas en las tiendas inteligentes ejemplifican cómo la Inteligencia Artificial y el Internet de las Cosas pueden unir, de manera efectiva, fuerzas para transformar el futuro de la venta al por menor. Con un creciente número de empresas proponiendo e implementando sus propios conceptos de tiendas inteligentes, como Amazon Go o Tao Cafe, un nuevo campo está claramente emergiendo. Debido a que las tecnologías utilizadas para construir sus infraestructuras ofrecen una importante ventaja competitiva, las empresas no están compartiendo públicamente sus diseños. Por esta razón, este trabajo presenta un nuevo modelo de tienda inteligente llamado Mercury, que tiene como objetivo mitigar la falta de información pública y accesible en este campo. No solo introduciremos un modelo general y completo de tienda inteligente, sino que también proponemos una implementación detallada y concreta para que cualquier persona pueda construir su propia tienda inteligente siguiendo nuestro modelo.

Palabras clave— tiendas inteligentes, tiendas sin cajeros, internet de las cosas, inteligencia artificial, comercio inteligente, autoservicio, comercio sin personal, fusión de datos de sensores, just walk out

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Concepts such as Artificial Intelligence (AI) or Internet of Things (IoT) are on the rise. Indeed, according to the AI Index Report for 2021 drawn up by the Stanford University, there were almost $50,000$ AI journal publications in 2018 [1]. On the other hand, there were over $8,600$ papers published involving IoT technologies [2]. Besides, the growth rate in the number of researches carried out has been drastically increasing in both fields during the last ten years.

We can put these disciplines along with their respective market sizes to better understand how these technologies are shaping the world. MarketsandMarkets estimated that the global AI market size will grow from USD 58 billion in 2021 to USD 309 billion by 2026 [3]. Likewise, according to the last industry market research report from Fortune Business Insights, the IoT market size stood at USD 250 billion in 2019 and it is projected to reach USD $1,463$ billion by 2027 [4].

Therefore, Artificial Intelligence and the Internet of Things are trendy fast-growing fields which will be making a huge impact on our lives in the near future. Though there are many social concerns that naturally arise when new technological breakthroughs come into play, we cannot dismiss the wide range of opportunities that these technologies have to offer.

With a market size of USD 21 billion in 2020, and an expected 195% growth by 2025, smart retailing is one of the greatest examples that show how Artificial Intelligence and Internet of Things could effectively join forces [5]. Despite this fact, even so there are quite a few papers introducing smart retailing concepts or ideas [6–8]; most of them introduce unmanned, unstaffed or cashier-less stores without diving deep into the technical issues that emerge, nor proposing solutions to overcome these problems. For instance, there is a clear lack of public research available to explain a feasible end-to-end implementation of cashier-free smart stores (with the only exceptions of the Grab [9], AIM3S [10] and 3S-Cart [11] models).

For this reason, we would like to introduce in this document a proof of concept of a partially automated store named *Mercury*. In this smart retail establishment, customers will be able to purchase products without being checked out by a cashier. They can enter the shop at any time and take whatever they want, leaving whenever they want without having to wait in a queue to check out. The store will use AI and IoT technologies to automatically compute the amount charged to each person.

1.1 What is a smart store?

A smart store is a retail store that combines smart devices[1] – such as smart shelves or smart carts – to reduce costs, enhance productivity and improve efficiency in both organizational processes and selling activities.

These two levels in which smart technologies for retailing are split up was already introduced in the literature by Eleonora Pantanoa and Harry Timmermans [15]. Concerning the organizational level, Pantanoa and Timmermans highlighted the potential of smart stores to implement technology to efficiently collect knowledge from customers (i.e. by codifying consuming habits) and transfer product knowledge into the service.

Conversely, regarding the selling activities, these establishments will be able to overcome traditional boundaries of physical points of sale as well as unlink the access to the product or service from the physical salesperson (i.e. unmanned or cashier-free stores).

In consideration of the foregoing, we will refer to smart stores in terms of a mercantile establishment that resonates with the smart usage of technologies for retailing purposes and further creates a smart partnership between the retailer and consumer following its adoption and enhancing real-time interactivity [16].

The introduction of smart retailing has already started in Europe with the self-checkout service, a system that is spreading and becoming more common within the continent [17].

On the other hand, in the United States, Japan and China, some companies are already implementing their own proofs of concept for smart stores which take a step further to completely remove the checkout line and automate the payment process. Some great examples of cashierless stores are Amazon Go by Amazon [18] or Catch and Go by NTT Data [19].

Figure 1.1: Self-checkout aims to speed up the checkout process in stores[2].

[1] There is a clear ambiguity issue with this term in the scientific literature that must be addressed [12, 13]. For this reason, a proper and concise definition for the term *smart* is required beforehand when introducing this concept. In this sense, we agree on the definition of smart devices provided by the SmartProducts Consortium [14].

[2] Source: PYMNTS.com `https://www.pymnts.com/unattended-retail/2020/self-checkout-hits-a-speed-bump/`

Figure 1.2: Amazon Go store[3].

The race is on to introduce these new technologies in a global retail market that is expected to reach USD 29,361 billion in 2025 [20]. Note that, notwithstanding the rise of e-commerce, brick-and-mortar networks will still account for over 80% of total retail sales (95% when restricted to grocery sales) by 2023 [21]. With big companies developing their own smart stores concepts, the future of retail will belong to the ones that can offer a true omnichannel experience.

Furthermore, these new automated or partially-automated retail establishments will lead to substantially cost savings and improvements in the customer experience. For instance, by shortening checkout queues or having more staff available to assist customers, some chief operating officers have reported up to a 12% in cost savings [21]. Additionally, an Adyen study revealed a potential sales lost of USD 37 billion due to long checkout lines. Moreover, they found out that 75% of the customers would shop more in-store with a just walk out payment experience [22]. This translates into a huge market opportunity that smart stores could likely cover.

1.2 Objectives

Companies are currently developing smart stores technologies to be able to materialize this new shopping concept in physical retail establishments. Some great examples are *Just-Walk Out* [23] or *Catch and Go* [19] technologies. Nevertheless, as we have stated before, though there have been a few papers published describing feasible implementations of these technologies, there are no public end-to-end descriptions available on how to integrate and combine smart devices to create these stores. For this reason, the technology is only accessible to big tech companies that are investing in this field.

Our main objective is to come up with a feasible infrastructure for a smart store to be able to operate, focusing on using devices and tools that are attainable and affordable. In this sense, our platform Mercury aims to take the edge off of the lack of public and accessible information, and research documents, in this field, by offering not just an example of how to build a smart store but a detailed infrastructure so that anyone can build their own system upon it. Besides, Mercury will be implemented using a modular approach to make it easy to scale up or update. Finally, although we will mainly focus on the problem modelling, we will also analyze the limitations of the system proposed.

Our own infrastructure proposal for the Mercury model will be conceived with a single shelf with two stands for products to be placed in. Clients will enter the establishment using a mobile app, being able to move around freely once in the store. Sensor fusion along with artificial

[3]Source: Amazon `https://www.amazon.com/b?ie=UTF8&node=16008589011`

intelligence engines will keep track of the interactions of the customers with the items in the store, automatically charging them on their bank accounts once they have taken their groceries and left the store.

In addition to the handling of events in the store generated by customers actions, we will also work on an explainability module which can provide human understandable explanations when required. Consequently, customers could request a clarification on why an item was charged to them when they receive their tickets after shopping. Furthermore, store managers will also be able to request explanations regarding the behavior of the system. For these cases, they will receive brief descriptions for the interactions and how they were assessed.

All in all, the assignment at hand is quite broad and ambitious. For this reason, the following subsections contain an itemized detailed explanation of the objectives as well as a list of tasks that needs to be completed in order to achieve them.

1.2.1 Specific objectives

In order to clearly specify the main objectives of this work, we enumerate them in this subsection:

[O1] Identify the main technologies and strategies used in the industry-leading smart stores companies to build cashier-less retail establishments.

[O2] Propose a feasible cashier-free smart store model using Artificial Intelligence techniques combined with Internet of Things devices. Put forward a simple implementation of the model to work under the assumptions of an uncrowded setup with little occlusion next to the shelves.

- [O2.1] Bring forward a set of sensors and smart devices to monitor the actions of customers inside the store, showing how sensor fusion can be implemented and highlighting edge cases.
- [O2.2] Avail of competitive and state of the art algorithms to fully leverage the sensors retrieved data and produce easy to consume metadata describing the scene, pointing up limiting cases.
- [O2.3] Design and implement an algorithmic solution to consume metadata describing the state of a store. The algorithm will be able to track customers in the establishment and decide who performed a certain action (as well as the action type – i.e. taking or dropping an item) when the shop state changes.
- [O2.4] Outline and apply mechanisms to generate human understandable explanations of the decisions made by the smart store when the state changes (e.g. justify why the store assigned a certain item to a customer after the item left the shelf).
- [O2.5] Handle real-time communication and interaction with customers in the store using a mobile app. The system will be able to notify any update on their virtual carts, as well as provide answers to requests made by clients (i.e. asking why a product was added to their virtual cart).

[O3] Suggest next steps for research to extend the model so that the risk of occlusion due to customers gathering is minimized. Explain how the implementation can be improved to operate in a production environment.

1.2.2 Tasks

For the sake of introducing the tasks linked to the specific objectives presented in the previous subsection, we will detail the tasks related to their respective main objectives.

Tasks related to O1

- Explore which companies are developing smart stores technologies. Study their particular smart stores concepts and extract patterns from them.
- Compile general strategies and technologies followed by industry leading experts to overcome issues when building these smart retailing establishments.

Tasks related to O2

- Analyze sensors and smart devices types from different companies in order to identify the ones that are useful for the model. Explore and choose attainable sensors or devices from the type required. Build own design and/or learn how to use existing smart devices. Investigate how sensor fusion can be applied to our case and dive deep into limiting cases that emerge from the setup.
- Study computer vision algorithms as well as algorithmic techniques to handle smart devices and sensors. Select and implement state of the art strategies to process the information received from the store sensors and devices. Propose a metadata format to simplify the process of consuming the data.
- Explore solutions proposed in the scientific literature to the problem of correctly identifying people performing actions such as grabbing items or moving around a room. Study the different existing tracking, object detection and pose estimation algorithms. Choose competitive strategies and algorithms to implement in the store.
- Research into explainability systems in artificial intelligence involving computer vision tasks. Implement a system that uses the metadata along with the video recorded to output a human-understandable explanation of the scene.
- Determine the mobile operating system and decide the tools which will be used to develop the application. Build and program the screens and logic required for customers to sign up, log in, check their virtual carts, enter or leave the store, as well as see their past shopping tickets. Allow customers to request clarifications on items purchased for their last ticket received.

Tasks related to O3

- Examine the model to find out how it could be extended to minimize the impact of occlusion. Study how computer vision experts address this problem.

1.3 Report structure

This document is divided into six different sections: Introduction, State of the Art, Mercury Smart Store model, A Mercury implementation, Scope, suitability and scenarios, and Conclusions.

In this first chapter, we have formally introduced the concept of *smart store* and we have outlined the main objectives of this work. We have also listed a set of tasks that will be accomplished and that suffices to achieve the goals exposed.

In Chapter 2, we take a gander to the state of the art of the technology used in the field of smart stores, as well as current realizations. We also spend some time diving deep into computer

vision related common issues and ways to tackle them down, with problems such as the camera placement dilemma.

Then, we move to Chapter 3, where we introduce the new Mercury Smart Store model. Mercury is a cashier-less smart store which can operate autonomously using sensor fusion and artificial intelligence techniques.

Furthermore, in Chapter 4 we cover the infrastructure we built to set up a partially automated cashier-less retail establishment following Mercury guidelines. We walk through the different modules that made up the system, from the smart shelves that monitor the items in the store to the mobile app which handles the communication with customers.

After that, in Chapter 5, we analyze the system and expose its behavior in real life scenarios, assessing the scope and suitability for different scenarios. We highlight the issues with edge-cases and study how to properly address limiting cases.

Finally, in Chapter 6, we conclude with the results obtained as well as provide next steps for future research.

Chapter 2

State of the Art

In the past few years, online retailing has risen to threaten the long-standing supremacy of brick-and-mortar retailers, although stationary retailing still prevails as an important interaction point for customers [24]. For this reason, most successful brands must learn to compete in both worlds. In fact, even big online retailing companies are building their own physical stores, sometimes integrated with their online ecosystem [25]. Therefore, the future of physical stores is to evolve using technology to join forces with online stores in a journey to improve the global shopping experience.

Besides, cashier-free shopping systems can help to improve the shopping process and, in spite of posing significant design challenges, there exists a bunch of companies which have given a try to develop their own solutions, that are nowadays working in production or as proofs of concept in the streets of countries like China, Japan, United States or the Netherlands.

Despite the fact that the aim of Mercury is not to replicate any of the already existing smart stores, their implementations are worth reviewing as they shed light on possible technologies or ideas to materialize the goals of Mercury. Therefore, in this chapter, we dive deeper not only in the different companies that develop solutions for smart stores, but also in the techniques they used to successfully accomplish their objectives.

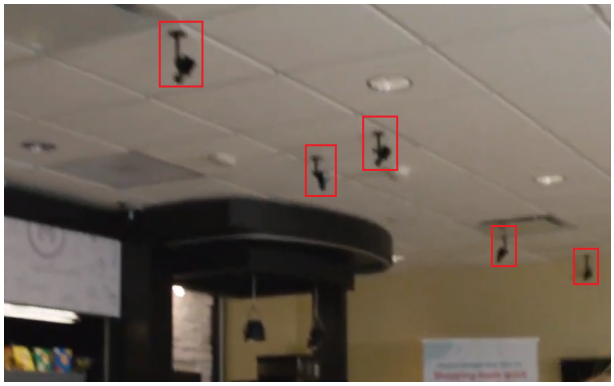
2.1 Technologies for smart stores

We study in this section what technologies were proposed by researchers to create this type of retail establishment, as well as the ones that industry leading companies claim to be using to build their own stores.

2.1.1 Smart cameras

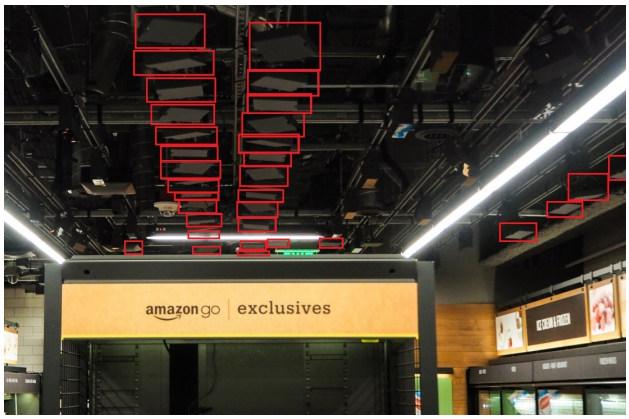
Smart cameras are embedded systems that can capture images and are capable of extracting application-specific information from these images, along with generating event descriptions or making decisions that are used in an intelligent and automated system [26]. Some common tasks that can be addressed with these cameras are online semantic segmentation, object detection, classification and facial recognition.

Processing images or video flows within the smart camera offers several advantages such as more autonomy to the system, lightening the workload of host devices, and/or prevent communication bottlenecks [27].



((a)) Standard store prototype cameras

((b)) Ahold Delhaize Lunch Boxes cameras



((c)) Amazon Go cameras

Figure 2.1: Examples of the extensive usage of cameras in smart stores

Thereby, smart cameras are worth considering when designing a smart store. Companies such as Standard or Ahold Delhaize are using alike devices for computer vision tasks (see Fig. 2.1(a) and 2.1(b)). Furthermore, even big retailing companies as Amazon introduced many cameras to track the events inside their Amazon Go stores (see Fig 2.1(c)).

All in all, the rest of this subsection is devoted to introducing cameras of particular interest when building smart retail establishments. The tasks related to computer vision and video analysis will be discussed in Section 2.2.

Internet Protocol Cameras

Internet Protocol Products, or IP Products, are devices that can receive control data and send information via an IP network. They are commonly used for surveillance, being IP Cameras the most relevant IP Products for our case of study.

Apart from IP Cameras, the most common surveillance systems are Closed-Circuit Television Cameras, or CCTV Cameras. CCTV Cameras transmit a signal to a specific place without allowing external connections; they were the dominant in the past with people not fully trusting IP Cameras despite they exceeded CCTV Cameras features and performance [28]. This trend was consistent with the fact that these cameras were more secure than IP Cameras, which were under continuous threats of external attackers. However, nowadays the technological advances have changed this pattern [29, 30]. For this reason, IP Cameras are currently one of the most common surveillance systems used and they are indeed far more secure than the old surveillance infrastructures.

Potentially, IP Cameras could be used to perform operations such as human tracking in the establishment or pose estimation of customers taking or dropping items (we will cover them at Subsections 2.2.5 and 2.2.6). Many of the IP Cameras available also offer these features integrated in the device, allowing the implementation of the distributed computing paradigm of edge computing in the stores.

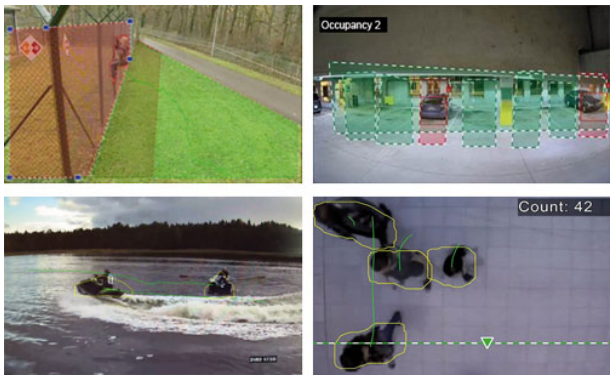
Bosch cameras

One of the most renowned manufacturers of security and smart camera devices is Bosch. There are many other companies developing products that are similar but we would like to introduce here the Bosch FLEXIDOME IP Starlight 8000i camera[1], which is the device that our realization of a smart store is prepared to work with.

The FLEXIDOME IP Starlight 8000i cameras are a range of smart, fixed dome cameras developed to highlight how the value of data is in the details. Besides, these devices also implement the industry-leading Intelligent Video Analytics[2] developed by Bosch, which allows users to gain insight into the events that the camera is recording or streaming. Among the main functionalities that Video Analytics provides are the Camera Trainer to train the IP device to identify objects while streaming video or the object tracker engine to monitor moving objects.

According to Bosch technical notes, intelligent tracking in their IP cameras is performed by using an optical flow based tracking algorithm for their IP moving cameras, while for fixed cameras their algorithm is based on Intelligent Video Analytics metadata [31]. The techniques based on Intelligent Video Analytics were developed through the usage of dedicated tracking models optimized for tasks such as indoor people counting or intrusion detection, and even though there is no public statement specifying the technology, they are likely to be based on optical flow analysis too [32].

((a)) FLEXIDOME IP Starlight 8000i camera



((b)) Region entrance detection (upper left), occupancy counting (upper right), tracking and moving object detection (lower left), and line crossing counting (lower right)

Figure 2.2: Bosch camera and Intelligent Video Analytics[3]

[1]FLEXIDOME IP starlight 8000i Datasheet: https://resources-boschsecurity-cdn.azureedge.net/public/documents/FLEXIDOME_IP_starlig_Data_sheet_esES_68669614475.pdf

[2]Technical Notes on Intelligent Video Analytics by Bosch: https://resources-boschsecurity-cdn.azureedge.net/public/documents/DS_IVA_7.10_Data_sheet_enUS_69630079883.pdf

[3]Sources: Bosch https://commerce.boschsecurity.com/es/es/FLEXIDOME-IP-starlight-8000i-2MP-X-series/p/80372299275/, https://commerce.boschsecurity.com/xl/es/Intelligent-Video-Analytics-7-10/p/69236903819/

2.1.2 Radio-frequency identification

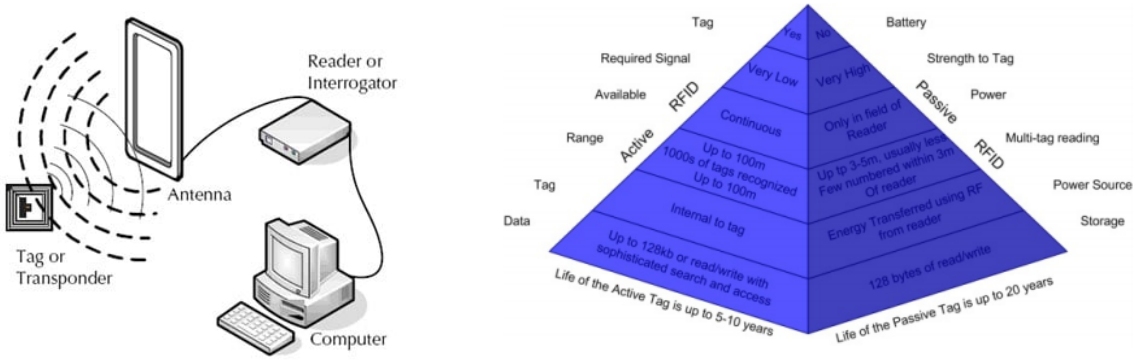
RFID (or Radio-frequency Identification) is a wireless communication way that incorporates the use of electromagnetic or electrostatic coupling in the radio frequency portion of the electromagnetic spectrum to uniquely identify an object, animal or person [33]. For this reason, RFID capability is really appealing as smart stores could potentially know exactly what their in-stock inventory is in near-real time, also tracking the quantity and location of goods.

RFID systems mostly work with tags attached to the items that are expected to be identified. These tags have their own ROM or re-writable internal memories in which the identification code is stored. These tags can be read by using different readers depending on the tag type.

There are three types of tags which are suitable to address different problems: active, passive, and semi-active/semi-passive. The main difference among them is that the semi-actives or actives ones require a battery, while the passives one take advantage of the signal received to power up the circuit. This means that while semi-active and active tags can have a more sophisticated memory system and work with very low signals in wide range environments, passive tags work only under restricted conditions of proximity with very high input signals and with a limited memory. On the other hand, the lifespan of the passive tags is sometimes four time longer [34]. A more detailed comparison among passive and active tags is shown in Figure 2.3.

Some well-known retailers have already incorporated the RFID mechanism to their supply chains. For instance, Walmart has improved its efficiency thanks to digitizing inventory information [35]. With accurate stock information available online, customers can see what items are currently in stock at each store and can also order out-of-stock items.

Moreover, in 2013, IBM presented a commercial showing the *future supermarket*, in which items grabbed from the shop were detected and charged to the bank account of the customer when leaving the store thanks to RFID tags [36]. However, it was not until 2018 that IBM opened a store in the UK where the checkout is carried by leaving the items in a specific surface that contains the RFID tags reader. After finishing placing the items, the bill is deducted from the bank account of the user. Figure 2.4 shows the surface for instant checkout.



((a)) Example of RFID tags working

((b)) RFID active and passive tags comparison

Figure 2.3: RFID tags working and types comparison[4]

[4]Sources: BarcodesEDGE https://www.barcodesinc.com/barcodesedge/guides/choosing-the-right-rfid-technology/, Cornell University https://arxiv.org/ftp/arxiv/papers/1002/1002.1179.pdf

[5]Source: Medium https://medium.com/@Jordan.Teicher/checkout-lines-are-so-2017-c3573e30b0e6

Figure 2.4: IBM's Five Second Checkout[5]

Another big company that studied and considered the usage of RFID tags is Amazon. Although there are no explicit statements explaining the inner working of their smart retail establishment, in 2015 the company filled a patent request describing an automated stock management system [37]. In that document, a new process powered by technologies such as RFID tags is portrayed.

Other retailers like Neiman Marcus are adopting RFID to deploy smart mirrors into dressing rooms [38]. These mirrors can automatically detect products that a customer is wearing via RFID tags and display those products as an image on the screen.

In consideration of the foregoing, RFID tags are a great technology worth reviewing when building a smart store, since they can directly target products taken by the customers and, therefore, make simpler the process of monitor them. Some companies as IBM have shown that they can be used in real life stores. Nevertheless, there are a few concerns that must be addressed when using this technology.

Although the price of these tags has been decreasing over time, and cheaper implementations are being proposed, they still range from 10 cents to 20 dollars [39, 40]. In a grocery store the range of prices for the products offered can vary greatly. For low-cost items of a price under 1 dollar, even the cheapest RFID tag would increase the price over a 10%.

Furthermore, this increment in the price of the items cannot be distributed evenly, as each individual product is charged. Hence, the overprice condition will always be held regardless of the performance of the store. This is not the case for other retailing strategies, such as having workers and cashiers or implementing a full automated infrastructure like the one presented by NTT Data or Albert Heijn. It is important to notice that the salary of the staff members will not vary drastically even if more items were sold; the same applies to maintenance cost of the infrastructure for smart stores that handle the automation without tagging products. For this reason, the potential profit margin could be greater by using other techniques such as computer vision and smart devices in the infrastructure, rather than tagging each and every product in the store.

2.1.3 Biometric identification

Biometrics seem to have a number of potential applications within retailing, including combating identify theft and fraud, increasing transaction speed at the point of sale, reducing queuing time at the checkout and transaction processing costs for retailers. Furthermore, it can also be used for information management, identification of employees, security in online retailing, and the development of more individually tailored marketing and customer loyalty programs.

Figure 2.5: Amazon One system[6]

For instance, as every person possesses distinct hand geometry, different techniques have been proposed for palm identification, specially as security concerns increases. Kong and Zhang were the first to investigate the orientation information of the palm for palm print and to propose a technique that gives almost 100% accuracy recognition rate [41, 42]. Nevertheless, despite the fact that biometic systems have the potential to identify individuals with a high degree of accuracy, it is important to notice that they certainly do not guarantee 100 per cent accuracy all of the time [43].

Indeed, palm recognition is the idea behind technologies such as Amazon One. Amazon One is an example of a biometric system that uses the information embedded in the palm to create a unique palm signature, and is capable of recognizing the palm in seconds, with no need to touch anything [44]. Figure 2.5 shows how the recognition process is carried.

This new technique is being tested by Amazon in 20 different stores such as Amazon Go, Amazon Go Grocery or Whole Foods Market [44]. What is more, Amazon offers other businesses to incorporate Amazon One to their establishments in order to provide customers a seamless service, faster payments, and a personalized experience.

Therefore, in the path of creating a quick and simple payment process, or identifying yourself when entering a retail establishment, the idea of performing these tasks with just the palms of the customers comes up naturally. As no two palms are alike and their features change little over time making it unique to customers, it opens a new horizon since it removes the necessity of having a mobile device to enter the shop. However, this idea assumes that every potential customer will have a readable palm print.

Conversely, face recognition can help resolve a wide range of issues facing modern day retailers, from ensuring security, to helping to understand customers and drive sales; the considered application of the right biometric technology can help to grow the profits and to deliver better customer service than competitors. Moreover, it is remarkable that some companies, like NTT Data, have already incorporated facial recognition at the entrance of their store [19]. This technique could potentially be utilized inside the store to bring a personalized shopping experience to the customers and perform analysis based on, for example, human emotion recognition.

On the other hand, although the application of biometric technologies may offer a number of benefits to retailers, there seem to be a great number of major barriers that will need to be overcome. In order to become an integral part of the retail scene, this technology will have to face barriers like acquisition and installation costs, doubts about the accuracy and performance

[6]Source: Amazon `https://www.aboutamazon.com/news/innovation-at-amazon/introducing-amazon-one-a-new-innovation-to-make-everyday-activities-effortless`

of biometric technology, and public concerns that its introduction will compromise personal privacy and civil liberties [43].

2.1.4 Smart shelves

Smart shelves can be conceived as the natural technological evolution of conventional racks. The most common practice to create this smart device is to use load/weight sensors to measure the quantity of the items available. Nevertheless, there have been some alternatives proposed that extend this model. Some of them are based on computer vision approaches to quantify if the product is in the right position (i.e. on the right shelf) or if it is placed with right visibility to the shoppers [45].

The main issue that smart shelves just based on weight sensors have to face is estimating the quantity and type of the item in the stand. While triggering an alarm when a product is removed is an easy to set up task, computing the items in the racks is not always realisable with just load sensors.

The alternatives proposed in the scientific literature for product recognition are mainly based on computer vision and pattern recognition, or RFID tags. However, there are many issues that need to be addressed to implement the first technique in retail stores [46]. For instance, items present a huge variability due to big catalogs, besides products appearance is changing frequently over time (e.g packaging design). Unfortunately, in-store items are sometimes really similar one to each other (e.g different flavors for the same brand of cereals). Even so, there are still some proposals that address these concerns, like the deep learning pipeline for product recognition proposed by Tonioni, Serra and Di Stefano [47].

As we have commented above, the other approach that researchers are considering includes smart shelves with RFID tags, RFID readers and antennas [48]. These smart shelves will collect customer generated data and automatically update their virtual carts. In Section 2.1.2 we cover in-depth this technology.

On the other side of the coin, some companies are starting to implement smart shelves not just for gathering information from customers, but to inform the clients. This is the case of the *Grab and Go* concept by NTT Data which offers dynamic pricing in their shelves, thanks to algorithmic approaches to estimate the value based on stock and demand [19].

Figure 2.6: Smart shelf with dynamic pricing[7]

[7]Source: NTT Data https://www.nttdata.com/jp/ja/-/media/nttdatajapan/images/news/release/2020/011600/011600-02.jpg?la=ja-jp&hash=379DC34E968F6BDE367806FA85109150594B05F2

2.1.5 Sensor fusion

Sensor fusion consists of the combination of data derived or retrieved from sensors in order to produce enhanced data in form of an internal representation of the process environment [49]. In particular, we call sensor fusion networks to the arrays of sensors. According to Durrant-Whyte [50, 51], there are three types of sensor networks configuration which characterize the sensor fusion networks:

- *Complementary.* When sensors do not directly depend on each other but can be combined to obtain a more complete picture of the event being analyzed (e.g. multiple cameras to observe disjunctive parts of a room [52]).
- *Competitive* or *redundant.* Each sensor delivers independent measurements for the same property. Visser and Groen [53] split up this case in two types: fusion of data from different sensors or fusion of measurements from a single sensor at different instants.
- *Cooperative.* The network processes data from two independent sensors to derive unknown information that would not be available from a single sensor (e.g. stereoscopic vision).

In smart stores is quite common to find all of these three types of configurations to solve different problems. Furthermore, there are smart stores concepts, such as the one introduced by Standard Cognition, which make use of an array of cameras as a redundant, cooperative and complementary sensor fusion network. In order to this, they placed cameras spread all around the store to obtain different views of the scene. Besides, these views overlapped so that they act as redundant or fault-tolerant networks (see Figure 2.8). Ultimately, by having two or more cameras, epipolar and projective geometry techniques can be applied to reconstruct the three dimensional space (see Figure 2.9).

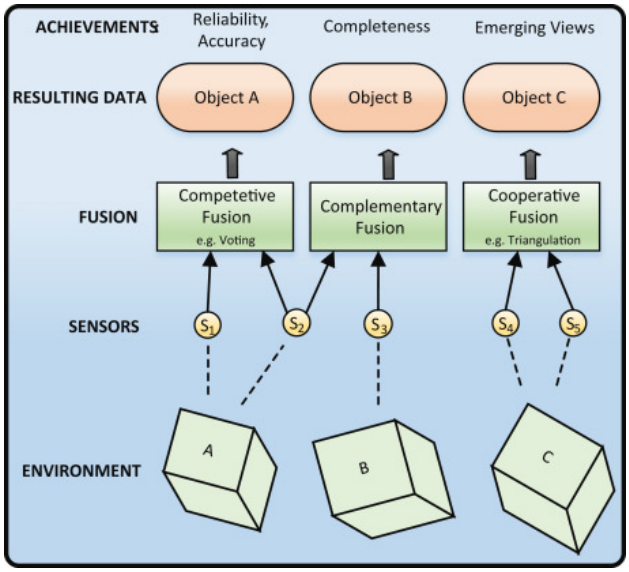


Figure 2.7: Sensor fusion networks configurations[8]

[8]Source: ScienceDirect `https://www.sciencedirect.com/topics/engineering/sensor-fusion`

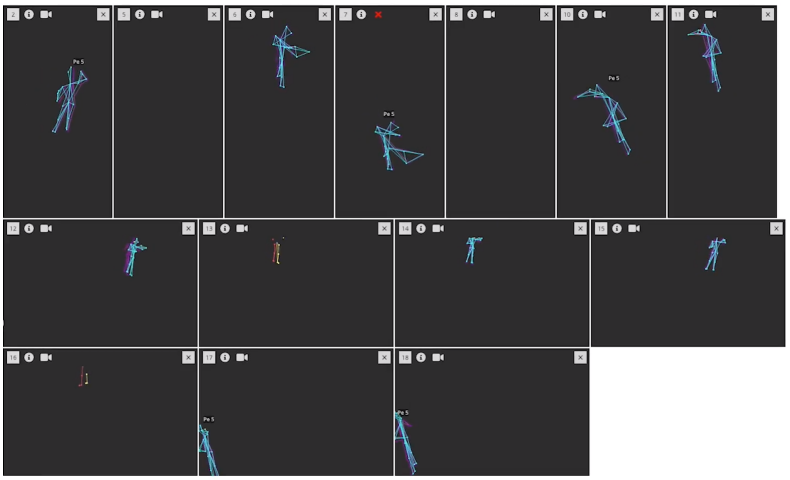


Figure 2.8: Sensor fusion network redundant and complementary view by Standard Cognition[9]

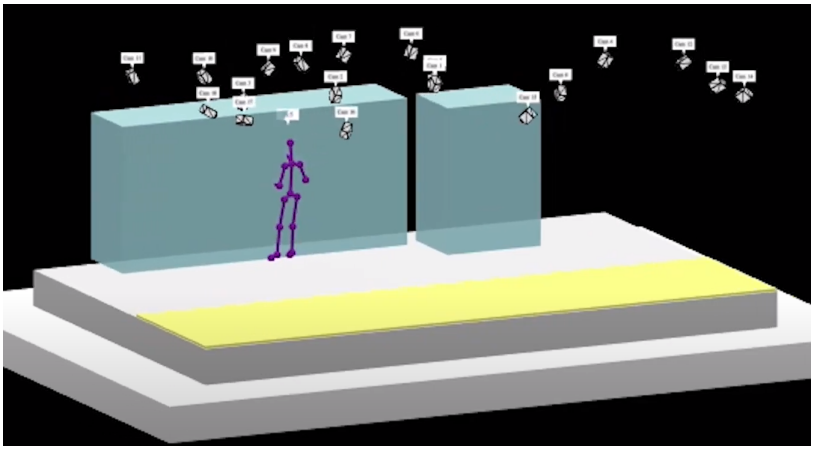


Figure 2.9: Sensor fusion cooperative network view by Standard Cognition[10]

Another great example of sensor fusion competitive network for smart stores are the introduction of RFID tags along with object detection. This is the case of Grab, a fast and accurate sensor processing technology developed for cashier-free shopping proposed by Liu, Jiang, Kim and Govindan [9]. This system uses a probabilistic framework to fuse readings from camera, load and RFID sensors to accurately assess who takes which product.

2.1.6 Recommender Systems

Recommender Systems (RSs) are software tools and techniques providing suggestions for items[11] to be of use to a client. The task of recommender systems is to turn data on users and their preferences into predictions of their possible future likes and interests. To this end, evaluations can be predicted or, alternatively, recommendation scores can be assigned to objects yet unknown to a given user. Objects with the highest predicted ratings, or the highest recommendation scores, then constitute the recommendation list that is presented to the target user [55].

Then, the usual classification of recommender systems is as follows [56]:

[9]Source: YouTube `https://www.youtube.com/watch?v=jppQoHUeEl4&feature=emb_title`
[10]Source: YouTube `https://www.youtube.com/watch?v=jppQoHUeEl4&feature=emb_title`
[11]*Item* is the general term used to denote what the system recommends to users [54].

- *Content-based recommendations.* Recommended objects are those with content similar to the content of previously preferred objects of a target user.
- *Collaborative recommendations.* Recommended objects are selected on the basis of past evaluations of a large group of users. They can be divided into:
  - *Memory-based collaborative filtering.* Recommended objects are those that were preferred by users who share similar preferences as the target user, or those that are similar to the other objects preferred by the target user.
  - *Model-based collaborative filtering.* Recommended objects are selected on models that are trained to identify patterns in the input data.
- *Hybrid approaches.* These methods combine collaborative with content-based methods or with different variants of other collaborative methods.

On the other hand, thanks to the ever-decreasing costs of data storage and processing, recommender systems gradually spread to most areas of our lives. In particular, in online shopping, sellers carefully watch the purchases of the customers to recommend them other goods and enhance both their sales and customer experience. However, brick-and-mortar retailers have yet to utilize this promotional strategy because of the difficulty of predicting consumer preferences as they travel in a physical space, but remain anonymous and unidentifiable until checkout. To yield this, recommender approaches by leveraging the consumer shopping path information generated by radio frequency identification technologies have been proposed [57].

Finally, we can notice that, although RSs have proven benefits for sales and customer experience [55], in general, smart stores are not yet including recommendation engines in their proposals (see Section 2.3). The only exception is the product recommendation included in Caper Cart [58]. Caper Cart recommendations are based on what customers have already added to the cart, so that they suggest products to complete previously known recipes by the system, as well as their location in the store.

2.2 Computer vision and deep learning for smart stores

Computer vision and deep learning techniques are present in most smart stores [9, 19, 23, 59], as they are really useful to deal with tasks such as assessing who takes an item, tracking customers inside the store or recognizing in-store products. In this section we cover the main algorithms for computer vision that are implemented by smart stores. We also discuss some problems that naturally emerge when dealing with image and video data.

2.2.1 The camera placement problem

Deciding where to place a camera or a set of cameras is critical for computer vision related tasks, as it has a direct impact on the quality of the data retrieved. In this sense, we first need to properly define the context of the problem we want to address. The solution would be different if we consider PTZ or fixed cameras, or if we have just one or multiple cameras. For this reason, we define the problem as it follows:

We refer to the camera placement problem as the problem of allocating a set of $n \geq 1$ fixed cameras in a three dimensional space, maximizing the coverage.

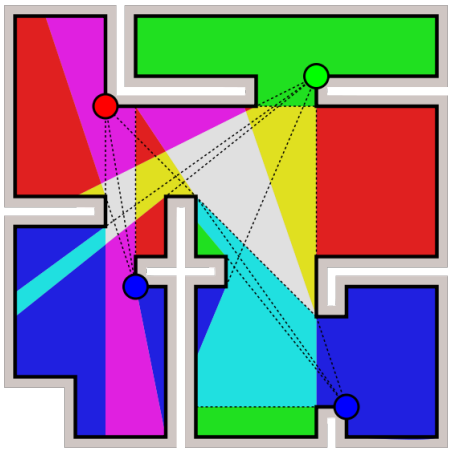


Figure 2.10: Example of a gallery where four PTZ cameras suffice to achieve complete coverage[12]

If we try to simplify this problem to a two dimensional space setup, we can use the results from the Art Gallery Problem studied by Chvátal in 1975 [60] so that we can express his results as it follows [61]:

Chvátal theorem adapted to 2-D camera placement.
$\lfloor \frac{n}{3} \rfloor$ cameras are occassionally neccessary and always sufficient to cover an $n-$vertex polygon.

Unfortunately, results from the plane cannot be easily extrapolated to the space. Besides, Chvátal theorem works under the assumption of fixed cameras that can see in any direction, which is not the case of almost every camera used in surveillance. Indeed, there are some issues that must be addressed to be able to apply an extension of this theorem, such as taking into account the limited view field, the effective range of the cameras, or the presence of moving or fixed obstacles limiting the line of sight in the scene.

However, there are some adaptations of the Art Gallery Problem approach by using implementations of the 3-Color algorithm [62]. Moreover, Pålsson and Ståhl proposed two strategies of camera placement within the rectangular algorithm which they called the basic and greedy approaches, in order to improve the results obtained from applying the 3-Color method [63].

Other approaches to the camera placement problem are based on multi-objective optimization to maximize visible coverage and minimize total costs under given jobsite constraints [64]. In this sense, the coverage of fixed cameras with no occlusion can be modelled by their visible distance and angle of view. In the case of spatial modeling it suffices to consider the width, length and height. If we consider a two dimensional space, we can discretize the scene into a two dimensional matrix as it is shown in Figure 2.11. This visibility analysis approach can be easily extended to higher dimensions.

[12]Source: Claudio Rocchini, CC BY 3.0 `https://creativecommons.org/licenses/by/3.0`, via Wikimedia Commons

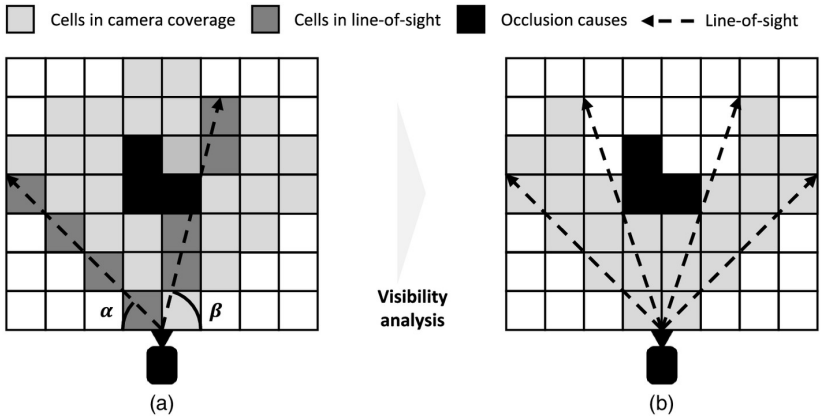


Figure 2.11: Visibility analysis over a discretized plane matrix with occlusion [64][13]

The single camera placement problem

If we restrict ourselves to a single camera placement problem, it suffices to maximize the only visibile area available. This optimization problem is quite simple, so we will now focus in different geometric considerations regarding the camera setup.

Assuming we are using a pinhole camera model [65], we can reconstruct the view from a single two dimensional plane projection of the three dimensional scene. Provided a camera C, we name **camera view** to the projection $\pi_C : \mathbb{R}^3 \to \mathbb{R}^2$ which resumes the computation of the image obtained by the camera.

Note that, in a single camera setup, we would not know the value of the points p or q in the three dimensional scene we are analyzing, but the value $\pi(p)$ and $\pi(q)$ of their corresponding projections. In particular, we can assume that the camera performs a projection of the space returning a two dimensional view of the scene.

In general, if we have a fix camera C with a projection π_C and we only know the values of the projection of two points $\pi_C(p)$ and $\pi_C(q)$, we cannot compute the value $dist(p, q)$ given by the Euclidean distance between the two points. In particular, when this is the case, we cannot get the values of p or q. An example that this holds can be easily constructed for two dimensional projections of points in a line, see Figure 2.12 where clearly $dist(B, C) \neq dist(D, E)$ and there are infinite points in the line from A to B whose distances to points in line A to C differ.

Note that the statement above for the plane is naturally extended to the space, as the first one is completely contained in the second.

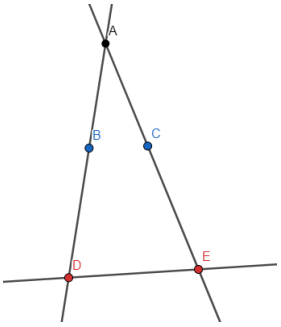


Figure 2.12: Projection π of points B and C with $\pi(B) = D$ and $\pi(C) = E$

[13]Source: ASCE Library https://ascelibrary.org/doi/10.1061/%28ASCE%29CO.1943-7862.0001636

In consideration of the foregoing, the main issue when projections are applied is, clearly, the impact of perspective in the view. In this sense, if we focus on orthographic cameras $C^{\perp}$ whose projections $\pi_{C^{\perp}}$ are orthogonal, then it is sometimes possible to retrieve the distance in the space using the two dimensional view. This holds since if we fixed two points $p, q \in \mathbb{R}^3$, there always exists a projection $\pi_{C^{\perp}}$ that is fully contained in any plane from the bundle of planes defined by the two points p and q (note that the orthogonal projection reproduces a plane view which can contain p and q, while preserving distances). Indeed, the following inequality always holds[14]:

Proposition. Let $p, q \in \mathbb{R}^3$ and $C^{\perp}$ be a fixed camera, then $d(\pi_{C^{\perp}}(p), \pi_{C^{\perp}}(q)) \leq dist(p, q)$.

Nonetheless, most of the time lens distort the image obtained, so we are not under the assumption of orthogonality. This means that the greatest the distance between p and q, the more inaccurate the results are. Despite of this, it is still possible to approximate the distance of the points by computing $dist(\pi(p), \pi(q))$.

As we have covered, even the simplest problem of computing distances using a single camera is complicated. This is not the case for all camera devices, but it impacts the most commonly used in surveillance. Indeed, there are great examples in the literature explaining how to create a spatial configuration based on a real-world environment by using stereoscopic cameras [66].

2.2.2 3D reconstruction from multiple images

Given a 3D scene captured by 2D images, we are interested in being able to reconstruct the three-dimensional scenario. The reconstruction is particularly useful since, when we are dealing with two dimensional projections, the depth knowledge is completely lost. In this section, we cover the main issues that arise when trying to reconstruct the three dimensional space using the camera views from a multi-camera setup. Besides, we also briefly cover some pertinent geometric strategies and algorithmic approaches.

The correspondence problem

The problem of determining which parts of an image correspond to parts of another image is known as the correspondence problem [67]. These two images can mainly differ due to the movement of the cameras and/or the movement of the objects. Correspondence is, indeed, a fundamental problem in computer vision that has been deeply studied due to its relevance for the field [68].

In particular, we are mostly interested in the correspondence problem for stereo vision, when the magnitude of the displacements between pixels in two images is generally called *disparity*, and it is directly related to the depth of the corresponding scene points [69].

Knowing the relative pose of the two cameras, we can consider the *epipolar line*, which is the straight line/ray through the optical centre of the image and image point in the other camera [70]. When two cameras are orthoparallel to each others, all epipolar lines are horizontal. Hence, the problem in the orthoparallel case can be conceived as a optic flow problem where the vertical

[14]The proof for the statement is quite simple. Note that, with no loss of generality, we can suppose that the orthogonal projection is over the plane described by the X and Y axes. Therefore, if we express $p = (x_1, y_1, z_1)$ and $q = (x_2, y_2, z_2)$ then $\pi_{C^{\perp}}(p) = (x_1, y_1, 0)$ and $\pi_{C^{\perp}}(q) = (x_2, y_2, 0)$. It is clear that now $d(\pi_{C^{\perp}}(p), \pi_{C^{\perp}}(q)) = \sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2} \leq \sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2 + |z_1 - z_2|^2} = dist(p, q)$.

displacement component is equal to zero. In the next subsection, we discuss more details about the mathematical approach using epipolar, projective and computational geometry.

Regarding the correspondence problem, due to the complexity of the task, some assumptions are usually taken such as brightnesss constancy (i.e the image intensities do not change under their displacement) [71]. Moreover, when dealing with different cameras placed at different locations, it is common to face issues such as partial left or right occlusion on the views.

Ultimately, although as we have briefly stated, the correspondence problem is quite complex, there are many feasible approaches proposed by computer vision researchers to tackle it down [72–75].

Computational, projective and epipolar geometries

For a setup of $n \geq 2$ cameras, it is possible to reproduce an stereoscopic view of the scene by combining epipolar and projective geometry (check Fig. 2.13).

Two perspective images can be used to fully describe a scene by using epipolar geometry. This relationship can be captured by a 3×3 singular matrix. If the intrinsic parameters of the centres of projections are known (e.g. focal length, coordinates of the principal point, etc) we can normalize the image coordinates, and the resulting matrix is known as the *essential matrix* (otherwise we use the fundamental matrix). This matrix contains all the geometric information for establishing correspondences between two images from which the three dimensional structure can be inferred [76]. One of the most well-known algorithms to estimate the essential or fundamental matrix is the Eight-Point algorithm. This algorithm was first introduced by Longuet-Higgins in 1981 [77], and it was later refined by Hartley in 1997 [78] who proposed a normalized version.

On the other hand, some issues arise when the extrinsic parameters such as the rotation and translation between the two images are unknown. However, this problem has already been deeply covered in the scientific literature [79, 80].

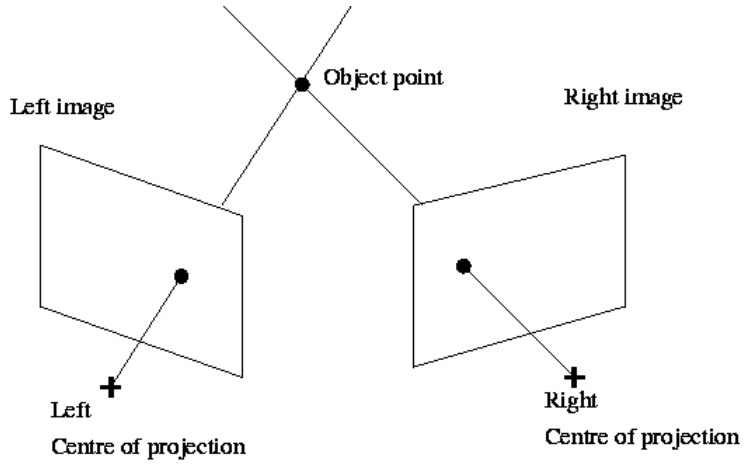


Figure 2.13: Triangulation example for stereo imaging[15]

[15]Source: University of Edinburgh https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT10/node3.html

After the scene is reconstructed, we can start applying computational geometry strategies to gather information. For example, we can apply techniques similar to the ones used to solve the Euclidean shortest path problem, which aims to find the shortest path between two points given a set of polyhedral obstacles that not intersect any of the points [81]. Besides, there are other interesting methods such as the Nearest Neighbor Search (NNS) or Locality-Sensitive Hashing (LSH) that can be effectively applied for computer vision tasks [82].

2.2.3 Occlusion detection and handling

Occlusion occurs in computer vision either when the object is hidden by the same type of object, which is called intra-class occlusion (e.g. in smart stores customer overlapping), or the object is occluded by a fixed element or an object of another type, this is called inter-class occlusion (e.g. a customer hiding a shelf) [83].

On the one hand, some studies proposed to deal with occlusion by creating datasets and training deep learning models [84]. However, because of the existing huge variability in object categories and instances, collecting and labeling a dataset with possible occlusions of each instance in every category is not feasible. For this reason, some researchers have proposed to rely on synthetic datasets or automatically generated samples [83]. These datasets could potentially be generated by using Generative Adversarial Networks (GAN) [85] or Conditional Generative Adversarial Nets (cGAN) [86].

On the other hand, there are other techniques, such as amodal recognition, which aim to perceive the scene when sensors are impacted directly by all of the elements. In this sense, there are two approaches proposed: moving embodied amodal recognition [87] and fixed amodal recognition (e.g. amodal instance segmentation [88]).

All in all, in the scientific literature, the problem of occlusion has been deeply studied. Most of the strategies previously used could potentially be applied to smart stores too. Techniques, such as the ones implemented to improve predictions for single-stage pedestrian detectors, can be modified to help customer tracking and detection inside smart retailing establishments [89, 90]. Besides, there are already extensions of state of the art object detectors such as YOLO to minimize the impact of occlusion [91, 92].

Figure 2.14: Results of Occlussion R-CNN and AmodalMRCNN models for amodal region segmentation [88] over COCOA [93][16]

[16]Source: Cornell University `https://arxiv.org/pdf/1804.08864.pdf`

Figure 2.15: Example of object detection for product recognition[17]

2.2.4 Object detection

The process of detecting instances of objects of a certain class within an image is named object detection. Detecting instances can be of great interest in areas such as video surveillance, self-driving cars, customer tracking inside a store or product recognition.

There are several models which perform the object detection task competitively with great accuracy levels. Some well-known approaches are Regions with Convolutional Neural Network features (R-CNN) [94], Fast R-CNN [95], Faster R-CNN [96], Single Shot MultiBox Detector (SSD) [97], Adaptive Training Sample Selection (ATSS) [98], Adaptively Spatial Feature Fusion (ASFF) [99], or You Only Look Once (YOLO) [100].

Stores such as Amazon Go probably implement object detection for their products inside the store [25]. This information combined with human tracking is critical since it can be used to identify customers who are taking or dropping a certain object. Moreover, it can also be useful to recognize items placed in the shelves. Although no object detection technique has been linked to any realization of smart stores, we decided to use YOLOv4 [101] in our proposed implementation for a smart store. YOLO allows us to get a competitive state of the art online object detector, pre-trained with people and common object samples which can be found in a supermarket (e.g. different bottles or fruit types).

2.2.5 Multitarget tracking

Being able to kown the real-time positioning of the customers is particularly interesting in smart stores. Human tracking is the task of estimating the location of a moving person through a set of consecutive frames of images. In general, performing these estimations is quite a complex labor and many algorithms have been proposed such as Compressive Tracking, Fragment Tracker, Multiple Instance Learning Tracker or Tracking Learning Detection [102]. Indeed, in the field, the problem studied is not limited to human tracking but it is generalized to multiple object tracking or multiple target tracking [103].

[17]Source: RetailVision `https://retailvisionworkshop.github.io/detection_challenge_2020/`

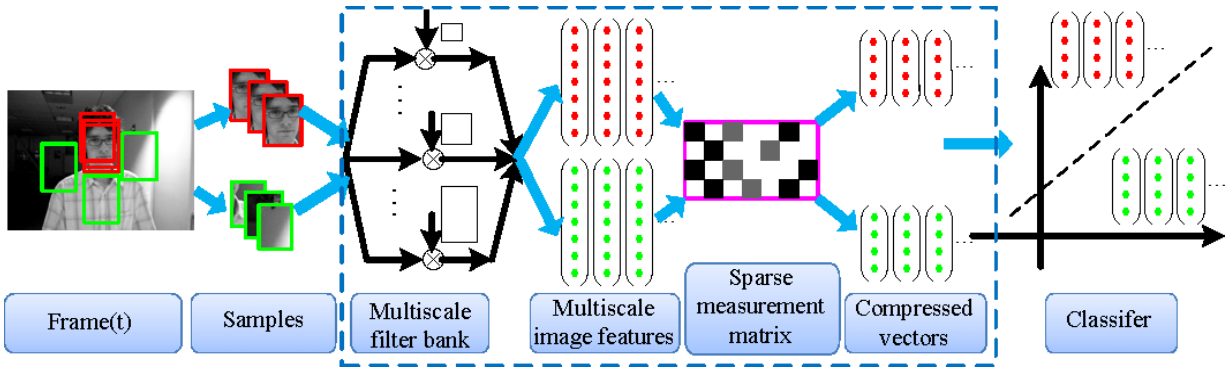


Figure 2.16: Compressive Tracking algorithm detailed[18]

When dealing with object tracking, there are many factors that can potentially impact the processing such as pose variation, illumination change, occlusion, or motion blur. Many algorithms address these issues by combining techniques that update their estimations with samples from observations in recent frames. One great example is Compressive Tracking, an appearance model based on features extracted from multi-scale image feature space with data-independent basis [104]. In particular, for this model, the tracking problem is addressed as a binary classification task via a naive Bayes classifier[19].

Although most smart stores models probably use computer vision and human tracking to perform calculations, the algorithms they used are not public [23, 105]. In our proposal for an unmanned smart store, we decided to use a state of the art tracking algorithms implemented using YOLOv4 [101] and the Simple Online and Realtime Tracking with a Deep Association Metric (DeepSORT) algorithm [106, 107]. Note that DeepSORT is based on the idea of tracking-by-detection, therefore YOLOv4 provides a solid foundation to build the system upon it (check Section 2.2.4).

DeepSORT extends the SORT algorithm [108], which implements a visual multiple object tracking framework based on data association and state estimation techniques. Although SORT did not solve issues such as occlusions or re-entering objects, DeepSORT addresses them effectively reducing the number of identity switches. In order to do this, the association metric from SORT was replaced by a more informed metric that combined motion and appearance information through the usage of a convolutional neural network [107].

All in all, tracking algorithms are really useful for smart stores since they offer a way of monitoring customers moving around in the establishment. This information can later be used to estimate who took or dropped an item to a shelf or notify when a user enters or leaves the store. Apart from helping to figure out who performed a certain action that changed the store state, people tracking provides really useful data for marketing analysis that can drastically improve targeted advertising.

2.2.6 Pose estimation

Given an image, human pose estimation is the process of determining the positions of human body parts such as the head, hands, wrists, knees or elbows [109]. This technique plays a major role for almost every smart store model. This estimation helps the computer to assess

[18]Source: The Hong Kong Polytechnic University `https://www4.comp.polyu.edu.hk/~cslzhang/CT/CT.htm`

[19]More information is available in the website published by the authors of the algorithm: `https://www4.comp.polyu.edu.hk/~cslzhang/CT/CT.htm`

the best candidate who performed an interaction with the store, such as grabbing or dropping a product.

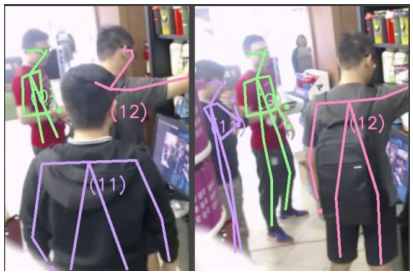


Figure 2.17: Grab system estimating the pose of customers[20]

Nevertheless, when dealing with a social scene, as it is the case for smart stores, the task of estimating the human pose is quite challenging. The visual appearance can considerably change depending on the camera perspective and illumination, as well as the pose or gesture of the person. What is more, large and persistent occlusions are frequent in crowded places, corrupting the image obtained [110]. We covered this problem in Section 2.2.3.

Although there are many issues linked to pose estimation, this method is still considered by many smart retailing models. Indeed, systems such as Grab [9] or Standard Cognition [111] are publicly acknowledging to be using it in their stores.

All things considered, there are many proposals on how to effectively implement this algorithm, many of them based on temporal convolutions and semi-supervised training [112]. However, the rest of the section focuses on the PoseNet model, since our implementation of Mercury architecture will make use of it.

The PoseNet model

PoseNet is a convolutional model developed by Google and it is included natively in TensorFlow [113]. PoseNet is based on the PersonLab model, which employed a convolutional network to detect individual keypoints and predict their relative displacement, allowing the grouping of points into a person pose instance. This model is competitive enough to beat most of the best state of the art pose estimation models [114]. For this reason, PoseNet was the model we considered to include in our realization of a smart store to estimate human poses.

Figure 2.18: Pose Estimation Example[21]

[20]Source: Cornell University https://arxiv.org/pdf/2001.01033.pdf

[21]Source: TensorFlow https://blog.tensorflow.org/2018/05/real-time-human-pose-estimation-in.html

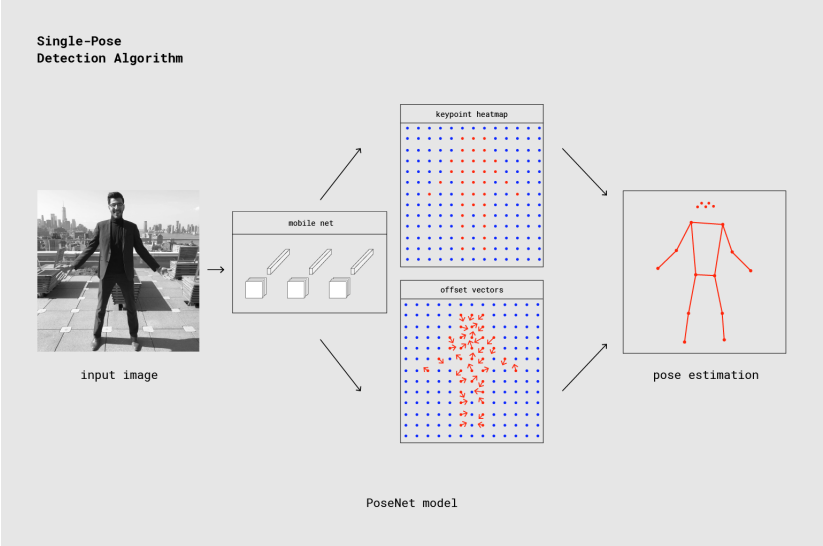


Figure 2.19: Single Pose Estimation overview in PoseNet based on Mobile Net[22]

The PoseNet model is trained over ResNet [115] and MobileNet [116], so there are two versions available. The ResNet based model provides higher accuracy at the cost of a larger size and extra layers that make the load and inference time greater. On the other hand, PoseNet distinguishes two operations to perform: single-pose estimation and multi-person pose estimation. In a smart store we are mainly interested in the second type of estimation, since single customer interactions can be estimated effortlessly. Nevertheless, the inner working of the multi-person computation is quite similar to the single-pose. The main difference is that, when there are more than one pose to estimate, the algorithm uses a greedy process to group keypoints into poses using displacement vectors along a part-based graph [114].

The single pose detector pipeline in PoseNet is shown in Figure 2.19. When processing an image, PoseNet returns a heatmap along with offset vectors that are decoded to figure out which are the confidence areas that match the pose keypoints. Conversely, in the heatmap, a confidence score is assigned to each position meaning that there is certain likelihood of a keypoint to exist in that position. By combining both outputs, the algorithm determines not just a keypoint confidence score for each human body part but also a pose confidence score.

Models, such as PoseNet, can effectively be used later to detect customers who are approaching objects with their hands to take them. For this reason, the information retrieved can be of great interest when discriminating who performed a certain action.

2.2.7 Explainability in computer vision

Before deploying a computer vision system, there is a strong need to validate its behavior; thus, establishing guarantees that it will continue to perform as expected when deployed in a real-world environment. In pursuit of that objective, ways for humans to verify the agreement between the model decision structure and their own ground-truth knowledge have been explored [117]. In this sense, Explanaible AI (XAI) was born as a subfield of AI, focusing on exposing complex models to humans in an understandable manner.

[22]Source: Medium `https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5`

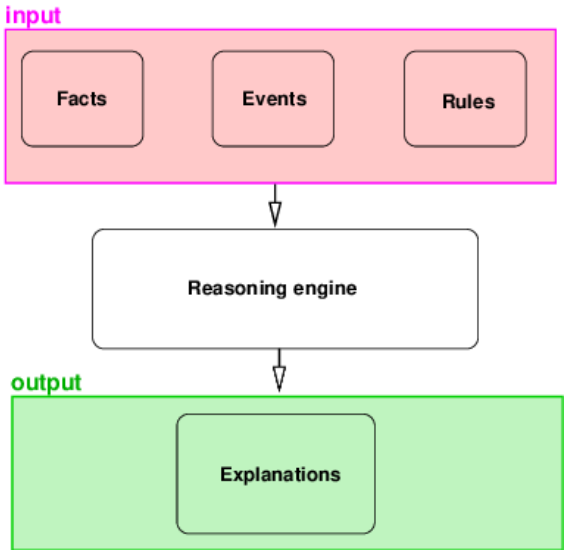


Figure 2.20: Reasoning process for explaining activity in video[23]

With the arrival of deep neural networks, this task started to become harder. In order to make predictions with a deep neural network, the data input is passed through many layers of multiplication with the learned weights and through non-linear transformations. Thus, a single prediction can involve millions of mathematical operations, depending on the architecture of the neural network [118]. Despite the complexity of deep learning models is a reason for their increased performance and accuracy, their nested non-linear structure makes them highly non-transparent (i.e. it is not clear what information in the input data makes them actually arrive at their decisions). Therefore, these models are typically regarded as *black boxes* [119]. Hence, while good performance is a critical required characteristic, explainability capabilities are highly needed to take computer vision to the next step and, in particular, include its techniques into decision support systems involving human supervision (e.g. security in a store).

For instance, in the case of pose estimation models, there is no explicit understanding on how the locations of body keypoints are predicted by convolutional neural network, and it is also unknown what spatial dependency relationships between structural variables are learned in the model. However, explanations can be introduced by analyzing the generated heatmaps, as Yang et al. did in TransPose model [120].

Moreover, researchers have managed to develop an automatic human behavior recognition and explanation model for CCTV video surveillance [121]. This is of potential interest because of its application on smart stores, not only due to the scenario of having static video surveillance cameras to monitor people interaction is presented also in retail establishments, but also because it compiles facts, events and rules – information that is easily obtainable thanks to the background knowledge about items positioning and shopping behavior – with the reasoning engine developed by Rigolli and Brady [122].

2.3 Smart stores companies

Since smart stores use really advanced technology which offers a competitive advantage, a detailed infrastructure has not been shared for any of these models we present. Nevertheless, for

[23]Source: Semantic Scholar `https://www.semanticscholar.org/paper/Behaviour-Recognition-and-Explanation-for-Video-Robertson-Reid/1f3309b177017b76cabab48f22f8107cfc137abe`

each of these stores, we compile all the available technical information that helps to get a grasp on the general techniques and engines used.

In the way of analyzing the available smart stores we realized they have common approaches, in the sense that they combine similar types of technologies to improve the performance of their systems. However, we noticed that payment is a part of the shopping process that allows us to classify into two different groups the existing stores: the ones that automate the payment process and the ones that require customer action.

2.3.1 Automated payment store models

Some companies are incorporating to their establishment automation on the payment process. Removing the necessity of carrying along cash or credit cards, in addition to directly exit the store once customers have taken their groceries, are some advantages that automated payment gives to customers. On the contrary, it adds the requirement of having an account on the particular company application or website of the shop the customer desires to enter, as well as detailing the bank information.

In the following, we present companies that are automating the payment process in their smart stores.

Amazon

One of the most well-known retailing companies is Amazon. In a way to innovation, in 2016, Amazon prototyped their idea of a smart store in Seattle and created the first Amazon Go store. The first establishment opened to employees on December 5, 2016 [123], and to the public on January 22, 2018 [124]. Since then, the model has been tested by thousands of customers and, by 2020, there were already 26 Amazon Go stores opened to the general public located in four different cities in the United States: New York, San Francisco, Chicago and Seattle [125].

The Amazon Go concept is designed to completely remove the checkout line and automate the payment process. Customers use the Amazon Go app[24] to scan a QR code which allows them to enter the store. Once in the store, customers can move freely and take or drop items in the shelves which are automatically added or removed from their virtual carts. Products in the virtual cart will be automatically deducted from the bank account of the customers once they leave the shop. Figure 2.21 shows these steps, that are also used in their two Amazon Go Grocery shops [126].

Figure 2.21: Amazon Go shopping process[25]

[24]Amazon Go app is free to download and available at App Store, Play Store and https://www.amazon.es/Amazon-com-Amazon-Go/dp/B01N2YE9FJ

[25]Source: Amazon https://www.amazon.com/-/es/b/ref=s9_acss_bw_cg_agojwo_1a1_w?node=20931384011&pf_rd_m=ATVPDKIKX0DER&pf_rd_s=merchandised-search-2&pf_rd_r=P55V23ZW6HZY0P8VEDJ3&pf_rd_t=101&pf_rd_p=21afee40-195f-4f61-a605-d0666e13ddb8&pf_rd_i=16008589011

Figure 2.22: *Just Walk-Out* Technology shopping process for non–Amazon retailers[26]

Amazon named the technology used for the Amazon Go store as *Just Walk-Out*, which they currently sell so that other retailers can implement their smart store concept in their own establishments [23]. The implementation of this technology for other retailers removes the step of login in a mobile app and scanning the code. Instead, customers have to enter using their credit card, as shown in Figure 2.22.

Amazon claims that *Just Walk-Out* was built leveraging the same types of technologies used in self-driving cars: computer vision, sensor fusion and deep learning [23]. According to Wankhede et al. [127], there are certain sensors that are potentially used by Amazon to implement this technology such as pressure detector/sensor, weight measurement, RFID tags and cameras. Sensors placed at shelves automatically detect when the product is taken from a stand or returned, and keeps track of the products in a virtual cart.

In a patent registered by Amazon, they proposed a sensor network to monitor the stock of a warehouse. Figure 2.23 shows how all of these various sensors are collaborating to track products. Although no public statement has been made by Amazon acknowledging that system is operating in the Amazon Go stores, there are remarkable similarities with the Amazon Go concept.

While Amazon Go was being developed as a concept, Amazon engineers also started to work in a different smart stores model. It was July 2020 when Amazon reinvented the shopping cart and introduced Amazon Dash Cart (see Figure 2.24), in which would rely the new Amazon Fresh smart stores. This cart includes a screen at the top where you can access your shopping list to check the items that have been introduced into it. It is also equipped with a coupon scanner and a weighing system [128].

The cart uses a combination of computer vision algorithms and sensor fusion to identify the items that customers put in it. When customers exit through the Amazon Dash Cart lane, sensors automatically identify the cart, and the payment is processed using the credit card linked to their Amazon account.

The aim of using this innovative shopping cart is to skip the checkout line and just roll out once the customer is done. However, Amazon Dash Cart does not have a great capacity as it only fits two grocery bags. It is specifically designed for small to medium-sized grocery trips.

At this moment, Amazon operates with 12 Amazon Fresh stores, that use this particular cart, in California and Illinois [129].

[26]Source: Amazon `https://justwalkout.com/`

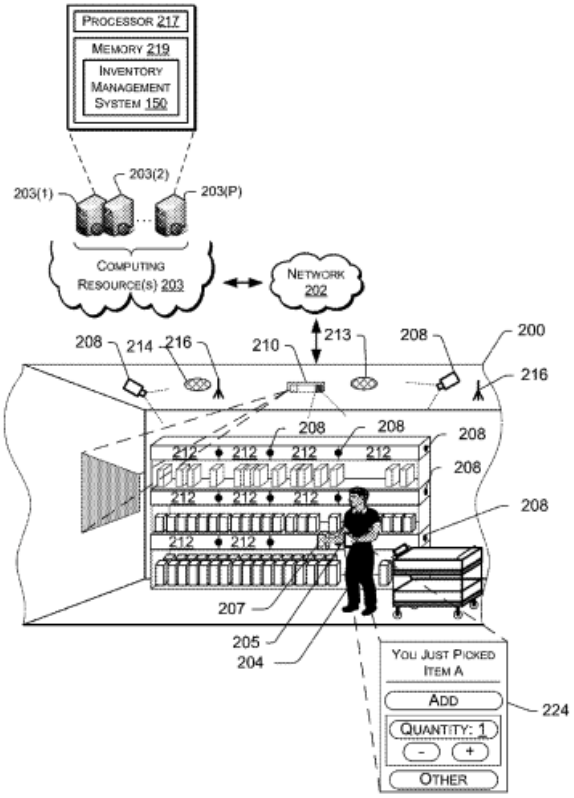


Figure 2.23: Patent for item identification and association presented by Amazon[27]

((a)) Amazon Dash Cart

((b)) Amazon Fresh shopping process

Figure 2.24: Amazon Dash Cart and its shopping process[28]

NTT Data

On September 2019, the firm NTT Data introduced their own experimental Smart Store named Catch & Go™. Moreover, the company plans to introduce this service at one thousand retail stores by the end of 2022 [19]. This store implements a facial authentication system which allows customers to register and enter the shop. The reason behind developing this feature is to avoid requesting customers to carry a mobile phone or a credit card with them while shopping. Therefore, customers are required to configure their payment method in advance. Figure 2.25 shows an example of the facial recognition entry.

[27]Source: Google Patents https://patentimages.storage.googleapis.com/5b/17/73/510f68b0ca9862/US20150012396A1.pdf

[28]Source: Amazon https://www.amazon.com/-/es/b?ie=UTF8&node=21289116011

Figure 2.25: Facial recognition at NTT Data Cashierless Store[29]

Inside the shop, there are a great number of cameras and sensors in the shelves that make feasible to recognize who has taken or left a product. Additionally, electric shelf labelling offers dynamic pricing functions, changing prices depending on the level of inventory. Their objective was to reduce waste loss and adjust reasonable prices based on demand.

Finally, once the customer is done, they can simply exit the store for the payment process to be carried, and they will receive the ticket in the NTT Data application.

Ahold Delhaize

Ahold Delhaize started testing another cashier-free store concept in 2019, named *Lunchbox*. These small-format establishments they are piloting, use frictionless checkout technology [130]. Figure 2.26 shows the existing store prototype.

The lunchbox store uses pose estimation and tracking systems to connect the right products to the proper shoppers, as skeletal tracking works well in small stores with 10 or 12 shoppers present at a given time [105].

This retail concept also completely removes the checkout lines from the establishment, allowing customers to exit the shop once they have taken their groceries and automating the payment process. Therefore, customers enter the lunchbox scanning their identification code and, after freely taking the desired products from the shelves, they can just walk out the store.

This smart store tries to make customers have a fast access for lunch groceries. That is why Ahold Delhaize expects to open these frictionless stores in offices, college campuses and airports [131].

Figure 2.26: Lunchbox pilot by Ahold Delhaize[30]

[29]Source: NTT Data https://www.nttdata.com/global/en/media/press-release/2020/february/announcing-the-introduction-of-facial-authentication-and-dynamic-pricing

[30]Source: Engage3 https://engage3.com/2020/03/ahold-lunchbox-frictionless-comparison/

((a)) Tao Cafe entrance

((b)) Tao Cafe exit

Figure 2.27: Tao Cafe store[31]

Taobao

Chinese e-commerce giant Taobao – belonging to the Alibaba group – presented in July 7th, 2017, an almost cashier-less cafe called *Tao Cafe* located in Hangzhou (China), in which they sell drinks and snacks as well as products such as backpacks, notebooks, plush toys, etc [59]. It is only opened to users of Alibaba's e-commerce site Taobao.

Taobao created a shopping process in which customers can scan a code in their mobiles at the door to enter the shop, grab what they need, and finally, when stepping out, the bill is automatically sent to their phones.

As we mentioned at the beginning, Tao Cafe, technically, is not entirely staff-less since waiters can take orders and baristas make drinks. However, for the purchase of drinks, a facial recognition-enabled camera system at the service counter automatically identifies the customer, links to their account, and processes the payment.

On the other hand, customers buying other products exit through a processing chamber equipped with multiple sensors, which identifies both the customer and the items they may have grabbed. Figure 2.27 shows the store appearance.

Sonae MC

The technology firm Sensei and the Portuguese food company Sonae MC, current owner of the brand Continente, inaugurated the first full-autonomous supermarket in Europe[32] in May, 2021, called *Continente Labs* [132]. In a similar way to the Amazon Go concept (see 2.3.1), this store completely removes the checkout line and automates the payment process.

Figure 2.28: Continente Labs shopping process[33]

[31]Source: High-Tech & Innovation Marketing Blogs `https://jakkimohr.com/2018/01/09/self-service-check-out-technologies-comparison-of-amazon-go-and-alibaba-tao-cafe/`

[32]Despite Ahold Delhaize introduced Lunchbox (Subsection 2.3.1) in the Netherlands, since it does not have a *supermarket appearance*

[33]Source: Continente Labs `https://labs.continente.pt/#/loja`

Figure 2.29: Continente Labs entrance[34]

In order to be able to enter the shop, customers scan a self-generated QR code from the Continente Labs app. Therefore, users must download the app and create an account. Once they have entered the shop, they can freely take or leave items from the shelves, which will be respectively added or removed from their virtual carts. When the customer leaves the store, they will see the receipt with the price breakdown displayed in their app, and it will be charged to their bank account [133]. Hence, it is necessary to register the bank information in the application before starting shopping. This process is shown at Figure 2.28.

At the moment this document is written, there is no information published about how Continente Labs store is designed. However, some media are reporting that the shop is equipped with 230 cameras and 400 sensors that the system uses to collect images and weight metadata, respectively, to detect the items grabbed or placed on the shelves [132, 134]. Furthermore, they apply computer vision techniques to assign the action to a specific customer.

An overview on the store appearance can be seen at Figure 2.29.

2.3.2 Manual payment store models

On the other hand, there are companies that prefer not including payment automation. Thus, they set up a specific part of the retail establishment as a self-checkout zone – where there are, either devices that recognize the products taken, or simply a self-service cashier to introduce cash or cards in order to pay for the registered products by the system – as it is the case of Decathlon stores, or put electric labels on the shelf that process the payment, as Albert Heijn does.

Decathlon

In 2019, Decathlon declared to have incorporated radio-frequency technologies in their self-checkout processes, although it was 2013 when they started installing RFID solutions to their shopping experience [135]. The main reason why Decathlon started to implement RFID was to increase inventory accuracy and avoid out-of-stocks. To fully benefit from the RFID deployment, they also use RFID tags on their products for loss prevention purposes. This enables Decathlon to track the merchandise end-to-end, from its arrival until the moment it leaves the store. Besides, retailers like Decathlon that use RFID tags expected to see their sales increase up to 5.5% [136].

[34]Source: Observador `https://observador.pt/especiais/entrar-por-os-produtos-na-mochila-e-sair-fomos-ao-primeiro-supermercado-em-portugal-como-a-amazon-go/`

Figure 2.30: Decathlon self-checkout[35]

In consideration of the foregoing, customers just need to introduce their items in the dark box (see Figure 2.30) that contains the tags readers that will identify the products grabbed as well as their quantities, and proceed to payment after that, although they can still use the traditional checkout.

Albert Heijn

Albert Heijn, one of the largest supermarket chains in the Netherlands, opened a new type of smart store called *AH to go* [137]. These stores implement a technology named *Tap to Go* which aims to reinvent the way we shop by allowing customers to tap with a card on shelf readers placed in the store to pay for the items. These card readers are placed in the shelves next to the items, so that customers can tap them and take whatever they want. Customers can also return a product without cost by dropping it back to the shelf and tapping the shelf reader again. The process of tap shopping is also introduced in Figure 2.31.

Albert Heijn claims that after extensive testing and improving on the store pilot, they are ready to process events in their stores. The goal of the company is to reduce to the minimum the average shopping time by removing obstacles for the customers. Though the first models used a card to tap, there are some stores that also admits readouts from the mobile phone. To process the payment, the system works by automatically withdrawing from the bank account of the customer the purchase price after 10 minutes of inactivity [137].

Figure 2.31: Tap to Go shopping process[36]

[35]Source: Cision - PR Newswire https://www.prnewswire.com/es/comunicados-de-prensa/openbravo-y-decathlon-refuerzan-su-asociaci%C3%B3n-con-el-despliegue-de-openbravo-pos-para-nuevos-mercados-en-asia-876850252.html

[36]Source: CM.com https://www.youtube.com/watch?v=Bf8bFKPAUdk

By 2021, there were over 45 AH to Go stores in Netherlands operating [138], though their expansions to other countries such as Germany had been troubled and delayed [139].

Figure 2.32: 17 Shandian smart store[37]

17 Shandian

17 Shandian is a staff-less convenience store. It uses monitoring and image recognition technology to stop theft, and customers must undergo a real-name authentication process to enter shops and pay by scanning codes. Stores are less than 20 square meters each and are usually located in office buildings and hotels to minimize inventory loss.

Since opening its first shop in June, 2017, the startup runs stores in Sichuan and Chongqing in southwestern China and boasts more than 100,000 customers [140].

Standard Cognition and Inokyo

Standard Cognition is a San Francisco-based startup that sells a technology based on machine learning and camera-enabled computer vision (i.e. pose estimation and tracking systems) [111]. They do perform object detection for the products and track each customer during their journey in the shop.

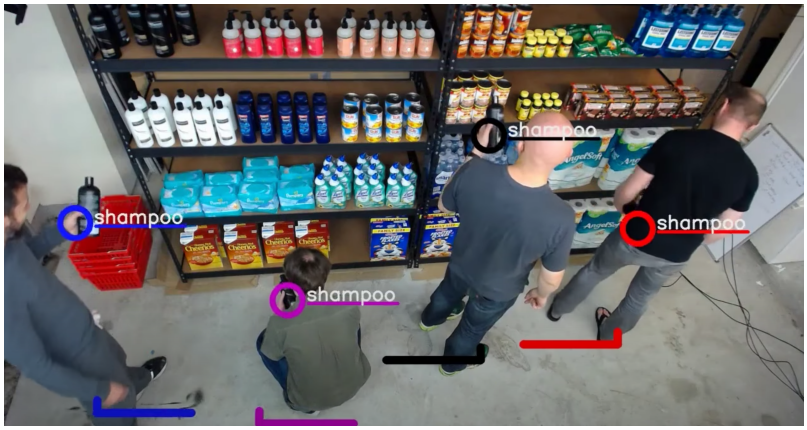


Figure 2.33: Standard AI recognition system[38]

[37]Source: Yicai Global https://www.yicaiglobal.com/news/staffless-convenience-chain-17-shandian-gets-a-round-funding

[38]Source: Rockset https://rockset.com/vision-ai-talk

Then, once customers have grabbed their desired items, they would need to go to the payment terminal to finish their shopping process.

Another company that sells this kind of solution is Inokyo [141]. Their system automatically detects new customers as they enter the store, without requiring gates, turnstiles or apps, and tracks the customer interaction with the store by using computer vision techniques such as object detection and human tracking. Once the customer is done, they just need to approach an Inokyo terminal and pay with their phone or credit card.

BingoBox and Auchan

BingoBox beginned its journey as a startup that opened its first outlet in Shanghai in 2017 after a few months of testing in its home city, Zhongshan, in southern China. It rapidly grown in a year by opening more than 500 flat pack-built stores.

Similarly, Auchan launched their own unmanned box named *Auchan Minute*, that has the same way of operating as BingoBox.

In both stores, customers enter by scanning a QR code with their phones, scan the desired products and proceed to payment.

((a)) BingoBox ((b)) Auchan Minute

Figure 2.34: BingoBox and Auchan Minute stores[39]

Figure 2.35: Famima store entrance[40]

[39]Sources: South China Morning Post https://www.scmp.com/tech/start-ups/article/2121799/bingobox-bring-unmanned-convenience-stores-hong-kong-next-year, China Daily http://www.chinadaily.com.cn/a/201804/19/WS5ad7f145a3105cdcf651931e.html

[40]Source: TimeOut https://www.timeout.com/tokyo/news/familymart-opens-a-self-checkout-convenience-store-with-no-cashier-040621

FamilyMart

FamilyMart opened in 2021 a self-checkout store with no cashier called Famima [142]. This store has just one employee stocking shelves, while computer vision techniques are applied to the data that over 40 cameras collect. This data helps to identify the items that customers grab as they move around the store.

The shopping process is similar to others presented in this section: the customer enters the shop, grabs whatever they desire, proceeds to payment and exits the store. Meanwhile, deep learning techniques are in charge of tracking the modifications on the store state.

Caper

In 2019, the firm Caper launched the first AI-powered shopping cart for grocery stores [143]. Caper Cart utilizes sensor fusion along with object recognition to know the items that are introduced in the trolley. One big advantage over its ceiling and shelf camera competitors is that this cart can not only promote deals on nearby or related items (see Figure 2.36), but it also shows a map detailing the position of the desired product. Besides, it adds recommendations based on what is in the cart to help customers fill out recipes.

Once the customer is done, the payment process is made by a conventional payment terminal incorporated in the cart as shown in Figure 2.36.

By the end of 2020 other shopping cart similars to Caper Cart arised. This was the case of Veeve smart cart[41] and Superhii smart[42] cart, that combine similar techniques and also require manual payment from the user.

Finally, Caper introduced in 2020 a counter for AI-powered checkout specially for small format stores [144]. The Caper Counter uses computer vision and sensor fusion technology to visually detect and instantly identify items placed on the Counter and automatically adds them to the total amount. It uses deep learning algorithms to visually identify every item from 5 different angles.

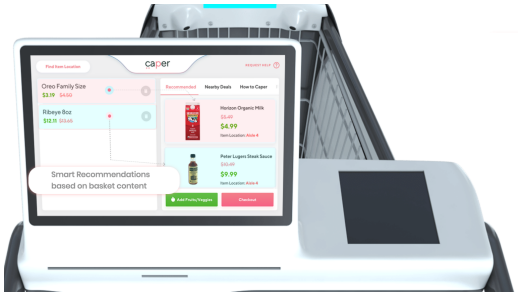
Imagr

In 2020, Imagr developed an image recognition solution designed to eliminate queues at check-outs. They created a smart cart that contains a code that customers will need to scan in order to start shopping. Once they have registered, they can freely move inside the shop and check their shopping status in a mobile application.

When the customer has finished grabbing their needs, they roll out to the cashier zones, where they will be requested to scan an app-generated barcode from which the products breakdown will be taken (see Figure 2.37) to proceed to payment.

[41] Learn more about Veeve cart at `https://veeve.io/#How`

[42] Learn more about Superhii cart at `https://www.superhii.com/superhi/news_d?NEWS_ID=b061b4e8da134583b3e395ee996bf0a6`



((a)) Caper Cart displaying the recommendation screen

((b)) Caper Cart payment

((c)) Caper Counter

Figure 2.36: Caper devices[43]

((a)) Imagr Cart

((b)) Imagr Receipt generation

Figure 2.37: Imagr Cart and receipt generation[44]

2.4 Conclusions

The implementation of a smart retail establishment is a complex task that usually combines many different smart devices through the usage of sensor fusion networks. These networks act in a cooperative, complementary and redundant manner.

The list of smart devices that are used by leading companies to create an array of sensors for later fusion sensor data contains smart devices such as smart cameras, smart carts, and/or smart

[43]Sources: Caper AI `https://www.caper.ai/`, Tech Crunch `https://techcrunch.com/2019/01/10/caper-shopping-cart/`

[44]Sources: Procreate Ltd `https://procreate.co.nz/imagr-smartcart`, Imagr `https://www.imagr.co/en`

shelves. These devices implement mechanisms such as RFID tags, optical sensors or load cells to effectively assess the action and the actor that performed a change in state of the store.

A particularly important technique for most of the smart stores concepts conceived is computer vision. Computer vision along with deep learning is commonly used to handle functionalities such as customer or product detection/tracking, or to retrieve more information from the clients with methods like pose estimation.

When dealing with image or video processing, there are many issues that arise. Occlusion, the correspondence problem and the problem of placing a camera to ensure a good coverage of the scene, must be addressed beforehand to guarantee high quality and accurate unprocessed image data.

Moreover, since many image processing methods make use of deep learning models, it is difficult to grasp a human understandable explanation of the behavior of the system. Explainability for black boxes models can be used in these cases to be able to deliver human understandable explanations for computer vision tasks.

Finally, it is important to mention that, though there are many manual payment smart stores model currently operating all around the world, there is a clear increment in the proposal and implementation of unmanned stores. Smart stores, specially the cashier-free ones, are becoming more popular and many of the proofs of concept that are operating today are heralding a new retailing era.

Chapter 3

The Mercury Smart Store Model

In this chapter we introduce the Mercury smart store model, explaining how the architecture was designed. Besides, we also detail the infrastructure required for the retail establishment to operate.

Mercury follows a modular approach to accurately assess which customer pick up which item, as well as it manages the communication with the customers to ensure they are notified when an object is added to their virtual carts. For this reason, we also include a section in which we detail each module.

3.1 Model overview

Mercury is our proposal for a cashier-less smart store that uses computer vision along with a sensor fusion complementary and cooperative network. In this section, we walk through the overall system and the modules which are required in order to operate properly. Furthermore, we explain the main functional specifications that are covered.

3.1.1 Architecture

In our architecture, there are several components which monitor the interactions of customers with the store through the usage of sensors and artificial intelligence engines. Each component is independent from the others, and the data they collect is sent to a main store server after being processed. The main purpose of this server is to coordinate the information received from other parts of the system and to manage the stock. As it was depicted, Mercury is implemented using a star topology with a hub which acts mainly as a communication orchestrator. Figure 3.1 represents this architecture.

Subsequently, we can outline three different types of modules in our model: modules that populate metadata generated by the data consumed from sensors or smart devices, modules that receive metadata to operate, and the hub.

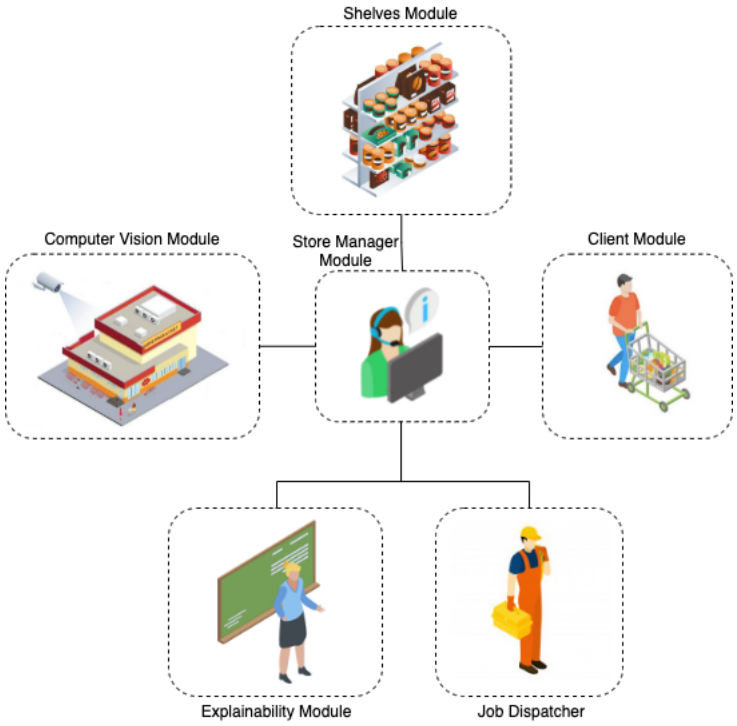


Figure 3.1: Mercury infrastructure overview diagram[1]

We named the hub in Mercury as the Store Manager (Section 3.2.4), and it is the component that handles and facilitates the communication among all the different units, as well as being in charge of the stock management. This module receives the metadata information generated by other components such as the Computer Vision Module (Section 3.2.2) and the Shelves Module (Section 3.2.1). Both of these units retrieve information from the environment and produce metadata, which is reported directly to the Store Manager.

When the orchestrator module has gathered all the metadata ready regarding an unknown event that changed the state of the store (i.e. someone took or returned an item to a shelf), it invokes the Job Dispatcher (Section 3.2.5). The Job Dispatcher is in charge of resolving which item was taken or dropped by which customer. For this purpose, this unit receives all the metadata available and starts a new job execution asynchronously to figure out who performed the action. Likewise, the Explainability Module (Section 3.2.6) is triggered whenever an explanation is required by customers or staff members. The Store Manager invokes this component by sending the metadata received for a certain time span, as well as the resolution provided by the Job

[1]Sources:
Client Module: VectorStock https://www.vectorstock.com/royalty-free-vector/shopping-people-3d-icons-set-isometric-view-vector-20695561
Shelves Module: dreamstime https://www.dreamstime.com/supermarket-food-shelf-icon-isometric-style-vector-web-design-isolated-white-background-image134036101
Job Dispatcher: Freepik https://www.freepik.es/vector-gratis/gente-profesional-trabajo-conjunto-iconos-isometricos_3887171.htm
Store Manager: VectorStock https://www.vectorstock.com/royalty-free-vector/call-center-operator-icon-isometric-3d-style-vector-7903189
Explainability Module: dreamstime https://www.dreamstime.com/classroom-foreign-language-teacher-icon-isometric-style-classroom-foreign-language-teacher-icon-isometric-classroom-foreign-image187341652
Computer Vision Module: 123RF https://www.123rf.com/photo_58367009_stock-vector-cctv-security-camera-on-isometric-illustration-of-supermarket-3d-isometric-vector-illustration.html

Dispatcher for the event. The main goal of the Explainability Module is to reshuffle the scenario to come up with a human understandable explanation.

Finally, the Client Module (Section 3.2.3) handles the notifications and communication with customers, such as keeping the virtual shopping cart updated or sending the purchase receipt to the client when they leave the store. It is also in charge of notifying when the customer first enters the store.

3.1.2 Functional specification

There are four common situations that are supported by Mercury. In this subsection we cover all of them by explaining how the different modules interact to address each case.

Customer entering the store

When a customer enters the store, the Client Module notifies the event. The Mercury model does not define nor force implementations to use any particular strategy to achieve this. There can be many feasible realizations such as including a QR code scanner (like some smart stores already do, check Section 2.3.1), or designing a mobile app that notifies the arrival (as it was decided for the implementation introduced in Section 4).

The notification is sent to the Store Manager, which will log this information, as well as invoke the Computer Vision Module so that it starts tracking the new customer inside the store. Note that any realization of Mercury does require to have a Computer Vision Module which can track customers inside the store. The process described is shown in Figure 3.2.

Figure 3.2: Interactions of the modules when a new customer enters the store

Customer performs an action

An action in Mercury is described as any interaction with the store that change its state (i.e. a customer grabbed or dropped an item to the shelves). Although both operations are different, the general process followed by the system to decide which customer triggered the action is the same.

Firstly, the Shelves Module notifies whenever a change in the shelves occurred. Then, any information that can be gathered by this unit is also sent to the Store Manager (e.g the weight or type of item). Then, the Store Manager sends a request to the Computer Vision Module in order to retrieve the metadata information available for the time span when the action was performed. Once all the metadata information has been gathered by the Store Manager, it triggers a new job execution in the Job Dispatcher to assess the situation. Once the situation is solved, the job results are sent back to the hub and then to the impacted customer via the Client Module. These interactions are fully depicted in Figure 3.3.

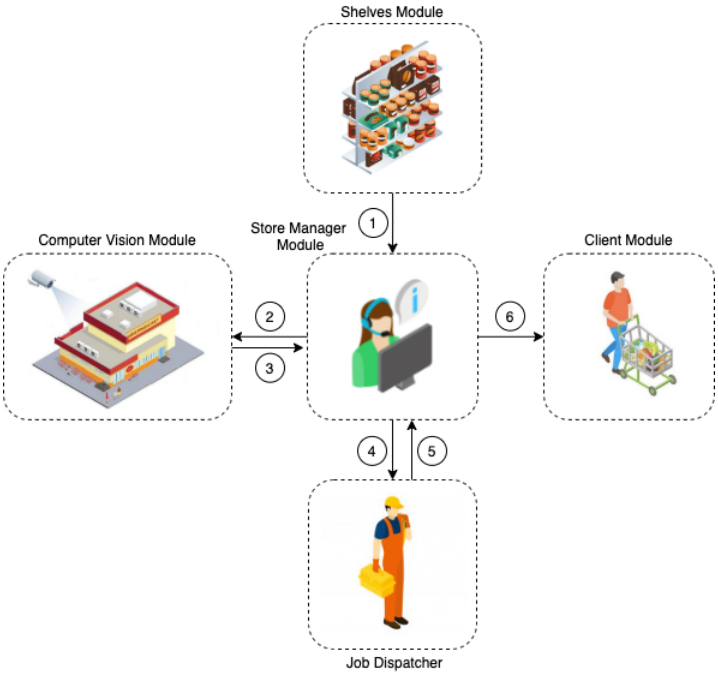


Figure 3.3: Interactions of the modules after a customer performs an action

Explanation requested

In Mercury, explanations can be requested by customers or staff. We first focus on the clarification requests that are made by customers, and then move to briefly introduce the explanations provided for the staff.

Any customer can request further details about why a product was added to their virtual shopping carts through the Client Module. This component is in charged of offering a simple way for customers to perform these requests (e.g. in our implementation proposal this can be done through a mobile application, check Section 4.4). The explanation request is forwarded by the Client Module to the Store Manager, which invokes the Explainability Module with the appropriate metadata to reshuffle the past scenario and come up with an explanation of what happened. The explanation generated is sent back to the Store Manager, and then to the Client Module so the customer can receive it. This whole process is shown in Figure 3.4.

Regarding explanations for staff members, any request can be directly made to the Store Manager Module to explain why the system behaved the way it did. In this sense, the clarification that is required differs from the one provided to customers. Staff members are mostly interested in the inner working of the system, as well as the decision-taking process in order to detect any issue, whereas customers are mostly interested in high level statements specifying when and why an item was charged to them.

Figure 3.4: Interactions of the modules after an explanation is requested

Therefore, the Explainability Module can also process requests made by the staff using the Store

Manager Module. These explanations correspond to the feature of the smart store to being able to explain itself.

Customer leaves the store

Whenever a customer decides to leave the store, the system must handle this event to act accordingly. Mercury handles the exit of every customer through the Computer Vision Module. When a customer exits the store, the Computer Vision Module will notify the Store Manager, which will compute a checkout ticket by invoking the Client Module and therefore keeping the customer informed.

The reason behind relying on the Computer Vision Module rather than another unit is to achieve a completely automated checkout process. The concrete implementation of this process can be attained by the usage of a special exit zone (like it is the case in many smart stores introduced in Section 2.3.1). Figure 3.5 covers the foregoing strategy.

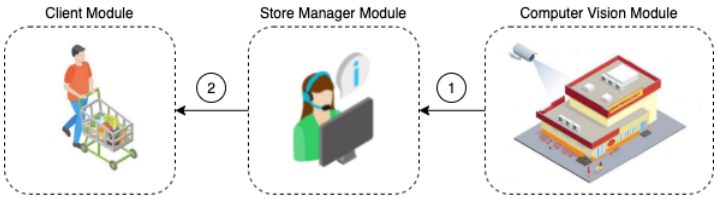


Figure 3.5: Interactions of the modules when a customer leaves

3.2 Modules in Mercury

When we overviewed the Mercury architecture, in Section 3.1.1, we introduced all the modules and some of their functionalities. In this section we expand the information for each module, detailing its functional requirements and presenting some implementation examples.

3.2.1 Shelves Module

The Shelves Module is the unit required to monitor the interactions of customers with the shelves. The main objective of the component is to be able to track items removed or added to any shelf. In this sense, we are not considering that an item could be removed/added to a shelf for any other reason than a customer interacting with the store. For this reason, the Shelves Module must be able to effectively detect and notify these changes. Once any of these changes take place, the Shelves Module will directly communicate with the Store Manager so that it can handle the situation.

Functional requirements

There are several strategies that can be followed to approach the recognition of changes in the shelves. However, regardless of the strategy followed, a proper realization of the Shelves Module must ensure that the next functional requirements are fulfilled (note that we list each Shelves Module functional requirement as ShM-Ri):

[ShM-R1] Identify when an item was removed or added, indicating the type of item, and the timestamp when the action took place.

[ShM-R2] Generate easy-to-consume metadata to express any action on the shelves.

[ShM-R3] Communicate the Store Manager any change in the state of the shelves.

Implementation proposals

There are several ways to implement the Shelves Module, including but not limited to:

- Monitoring the shelves using computer vision. This is indeed the technique implemented in smart store models such as the one introduced by Grab [9].
- Force transducer sensors in the stands. In Section 4.2 we expose a concrete and feasible implementation using load cells as weight sensors.
- Smart shelves based on RFID technology. As we introduced in Section 2.1.2, RFID systems can operate using RFID tag scanners or alike devices to track the stock in the stores.

3.2.2 Computer Vision Module

The Computer Vision Module does not only record the video streaming data, but it also pre-processes the information to generate easy-to-consume metadata for the Job Dispatcher and the Explainability Module. In this section, we describe the functional requirements as well as propose some feasible and attainable implementations.

Functional requirements

There are several ways to implement the Computer Vision Module, being the only one mandatory requirement to have a device to record videos or images of the store. Nevertheless, any realization of this component must suffice the following functional requirements (note that we list each Computer Vision Module functional requirement as CV-Ri):

[CV-R1] Uniquely identify and track customers inside the store.

[CV-R2] Detect actions of a customer towards performing a change in the state of the store (e.g a customer taking an item from a shelf).

[CV-R3] Record a video of the moment an action was performed.

[CV-R4] Generate easy-to-consume metadata exposing the state of the store before an action was performed and after the action was performed (e.g customer poses, customer positions relative to the shelves).

Implementation proposals

We include in this section some of the most important features that should be included in the component, as well as some examples to implement them. Note that the list shown is not comprehensive, and it just aims to highlight technologies and implementations worth considering:

- Identity tracking. This is almost mandatory to meet the requirement [CV-R1]. There are many ways to tackle identity tracking, but most of them use deep learning and algorithmic approaches. One great example is the DeepSORT algorithm [106, 107], but there are also some interesting approaches such as the ones based on color histogram tracking to reduce occlusion issues [145].

- Pose estimation. There are already some models that acknowledge to use skeletal and pose estimation such as the smart store introduced by Standard Cognition [111]. Besides, there are models available that effectively implement this technique such as PoseNet [114] or OpenPose [146].
- Product recognition. To support and enhance the Shelves Module, the Computer Vision could implement its own product recognition engine. However, as we have pointed out in Section 2.1.4, there are some problems linked to this approach such as product variability and training complexity of deep learning models. Nonetheless, as we have already stated, there already exists models proposals to address these concerns [46, 47].

3.2.3 Client Module

The Client Module owns the overall infrastructure needed to handle the communication with the customers. It is in charge of the registration of the customers, keeping the shopping status up-to-date by adding or removing products from their virtual carts, as well as handling possible explanation requests that they may require. In this section, we cover the functional requirements for any Client Module to operate, as well as introduce some feasible implementation proposals.

Functional requirements

In order to ensure that the Client Module can operate as expected along with other Mercury components, the following requirements should be met (note that we list each Client Module functional requirement as CM-Ri):

[CM-R1] Enable a way for customers to register and notify their entrance to the shop.

[CM-R2] Allow customers to check their current shopping status.

[CM-R3] Send the shopping ticket to the customer once they leave the store.

[CM-R4] Allow customers to request a clarification on any product assigned when leaving the shop.

Implementation proposals

In order to successfully address the functional requirements exposed for the Client Module, there are many reasonable approaches. However, we now introduce a non-comprehensive list of examples that can lead to the materialization of this module:

- Login handling. There are many ways to manage the register and login of customers in the store, including but not limited to:
  - Facial recognition. This approach is used by already operating smart store concepts such as the Catch & Go technology by NTT Data [19].
  - Palm recognition. Another biometric technique previously studied to implement in smart stores is palm recognition. There are smart stores that are already using this technique, like Amazon One in the Amazon Go stores [44].
  - Logging in with a mobile app. This is the case that we cover in our implementation (see Section 4.4). The mobile application can be used to expose a QR code which allows customers to enter the store through a QR code flap barrier gate.
- Customer notification. A mobile app could be a great approach to manage the communication with the customer, but there are other methods such as non real-time communication through mail messaging.

3.2.4 Store Manager

The Store Manager is the component that orchestrates the communication among all of the modules. It can be conceived as a centralized manager hub to redirect information accordingly to the demand. Besides, the Store Manager owns the store database access, and, for that reason, is on charge of managing the stock. In this section we go through the main functional requirements that must be fulfilled by any implementation of the Store Manager. We also comment some feasible approaches which can be used to build a store manager.

Functional requirements

Implementing the Store Manager is quite complex, and we must ensure that there are several functional requirements met. For this reason, we now list the main features that every implementation must consider for a successful deployment of Mercury (note that we list each Store Manager functional requirement as SM-Ri):

[SM-R1] Oversee and administer the communication among all of the Mercury components by redirecting information when required.

[SM-R2] Formulate requests on demand to inform properly of store changes to all the actors involved.

[SM-R3] Manage the stock and database of the store.

Implementation proposals

The Store Manager must be able to coordinate the whole system. There are many feasible implementations of the module. However, it is worth considering that the component must be able to communicate with the other units. From the Mercury architecture, the star network topology for a client-server setup is naturally derived. However, Mercury does not restrict to this setup, although it is encouraged. The handling of communication can be also implemented following different protocols or techniques. For instance, in our implementation we will introduce a Store Manager server which communicates via TCP/IP socket connections with the other units (see Section 4.5).

3.2.5 Job Dispatcher

The Job Dispatcher is in charge of assessing which item was taken or dropped by which customer whenever the state of the store changes. For this purpose, this module receives all the information available from the Store Manager Module and starts a new job execution asynchronously which will figure out who took or dropped the item. In this section, we will go through the main functional requirements that, when addressed, support a successful implementation of the unit. Besides, we discuss some proposals on how to implement the module.

Functional requirements

The main objective of the Job Dispatcher is to decide who performed a certain action that changed the state of the store. With the aim of achieving this, the Job Dispatcher uses metadata produced by other Mercury units such as the Smart Shelves or the Computer Vision modules. All in all, the following functional requirements (Job Dispatcher functional requirements, JB-Ri) must be met by any implementation of the component:

[JB-R1] Consume metadata generated by the Computer Vision and Smart Shelves modules.

[JB-R2] Assess every candidate using metadata and grant them a score to evaluate the likelihood of being the one who performed an action.

[JB-R3] Communicate with the Store Manager the results of the analysis of a batch of metadata.

[JB-R4] Process concurrently simultaneous metadata batches for different cases.

Implementation proposals

There are many ways to implement the Job Dispatcher, but the implementation itself mainly relies on the metadata available. In this sense, there are several different strategies followed by some companies or proposed by researchers.

One great example is the probabilistic assignment framework developed by Grab, in which they leverage the information from cameras, weight sensors and RFID receivers to determine the likelihood that a given client picked up a certain item [9].

Nevertheless, there are many other examples such as the Amazon patent [37] describing an automated and smart managed warehouse, or the techniques allegedly used by Standard Cognition in their stores [111].

3.2.6 Explainability Module

The Explainability Module is responsible for generating high level, human-understandable, explanations on demand. In order to accomplish this, the component receives metadata regarding a change in the state of the store and a certain job results, generated via the Job Dispatcher, from the Store Manager.

In this section we take a look at the main requirements for implementing the Explainability Module following up Mercury guidelines. We also devote a subsection to cover some proposals for the implementation of this unit.

Functional requirements

As we have already stated, the Explainability Module must be capable of providing high level explanations of what happened in the store. There are several concerns that arise when dealing with these tasks, such as defining a proper format that is easy to understand for humans. Furthermore, there are different user types that can ask for explanations such as staff or store managers.

With this in mind, the following functional requirements (Explainability Module functional requirements, EM-Ri) must be achieved for a successful realization of the component:

[EM-R1] Generate tagged videos that might contain metadata information that is easy to understand for customers.

[EM-R2] Protect the privacy of other clients when providing explanations on demand.

[EM-R3] Maintain a database of past cases and re-utilize the knowledge acquired to answer explanation requests from staff or store managers.

Implementation proposals

By the time of writing this document, there are no smart store models providing high level explanations to their customers. Besides, it is unknown if Amazon Go, NTT Data stores, Taobao Cafe, or any other smart store, are generating explanations of the model to the staff.

For that reason, we cannot introduce some examples of past successful implementations of explainability techniques to this problem. Nevertheless, there are many feasible approaches that can be considered when dealing with explainability. For instance, to explain the system itself, as we have already stated in Section 2.2, there are techniques already proposed to address explainability in deep learning algorithms and models such as OpenPose [146]. Moreover, discretizing the past cases, and storing the information appropriately, an approach based on the 4R of the Case-based Reasoning cycle [147] can be used. Indeed, this is the methodology followed in our implementation of the Mercury model (see Section 4.7).

On the other hand, clarifications for customers can be obtained by recording the exact moment when an action was performed, and delivering the video when the explanation is requested.

Finally, it is also important to notice that explanations can differ drastically from one user to another. For instance, customers mainly want to know why the system added a certain item to their virtual cart. This information can be useful if they want to double-check what happened and notify any issue or ask for a compensation (e.g. a video explanation received by a customer does not show that the customer took the item but other customer did). Conversely, staff and store managers are mainly focus on understanding the inner working of the system. This means that the clarification they receive must be useful to detect or fix any potential issue (e.g. after receiving several complaints, the store manager requested a clarification which exposed that the limit of people in store was exceeded, critically impacting the predictions made).

3.3 Conclusions

As we have covered in this chapter, although Mercury aims to serve as a straightforward model of an unmanned cashier-less smart store, there are some complex situations that must be addressed. In this sense, the model proposed establishes the main guidelines and features required for a proper smart store to operate autonomously and cashier-free. The easy-to-build modular infrastructure covers most of the cases that occur in a store, and can be easily extended to match expectations.

By relying on different components to gather metadata and sensors data, as it is the case of the Shelves and Computer Vision modules, the system can fully focus on each task by running dedicated job instances on demand. Whenever a customer changes the state of the store (e.g. taking or dropping an item to a shelf), the Job Dispatcher is able to use the metadata information generated by the Computer Vision and the Shelves modules to assess the best fitted candidate.

Regarding the communication with the customer in Mercury, it is completely delegated to the Client Module. This component handles the notifications sent and owns the relationship with clients.

Conversely, the communication among all the modules is orchestrated by the Store Manager. This module does not only redirect information when appropriate, but it is also capable of performing requests to other components when more information is required for the system to operate.

Finally, when they are required, explanations on the predictions can be provided by the Explainability Module. This module does not only reshuffle the scenario to reproduce the events and highlight the important information for customers, but it can also clarify past issues to the staff.

On the whole, the Mercury concept exemplifies how a cashier-less smart store can be built using a concrete approach which covers most common scenarios that occur in a store.

Chapter 4

A Mercury implementation

We introduce in this section a proof of concept implementation of the Mercury model. Besides, we identify some complex situations that are left out of the scope of this work, as well as clearly state the simplifications considered for our model. Moreover, we explain how we integrated smart devices along with sensors to build the sensor fusion network. In the following chapters, we also detail the response of Mercury to certain cases and scenarios, and propose extensions to it.

4.1 Implementation overview

In order to provide an example on how the infrastructure proposed for Mercury (Section 3) can be implemented, we are introducing our own implementation. A feasible approach to implement the Mercury model is shown in Figure 4.1.

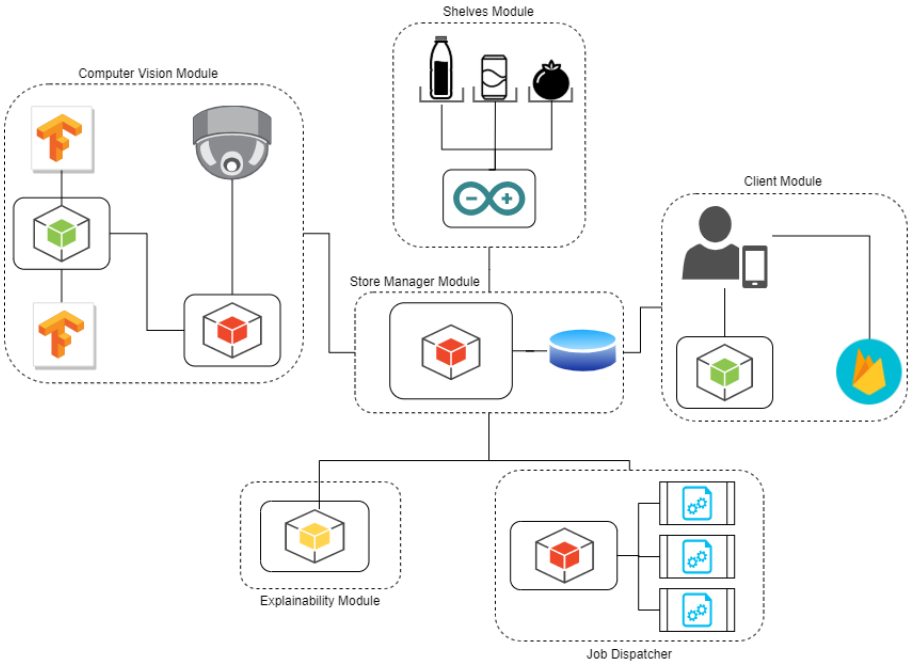


Figure 4.1: Mercury own infrastructure implementation overview diagram

The diagram includes the main technologies that we decided to use in our realization of Mercury

(all of the modules will be studied in-depth in their corresponding sections). Furthermore, we include in Section 4.1.3 some notes on the main development concerns related to our proposal. Finally, we describe the main tools used to develop and deploy Mercury for testing purposes, as well as the platforms that are supported.

4.1.1 Simplifications

The environment in which commercial establishments operate is complex and many unexpected events might occur that staff members should overcome. Smart stores must deal with these situations solving them by applying automatic or semi-automatic rules, along with the intervention of shop employees. For instance, if a customer breaks a product inside the store, how should the system respond to the event? The shop might automatically charge the price of the item to the customer or a store clerk must step in. There are a lot of different scenarios, like the one presented before, that need to be considered before delivering the smart store model to production.

In this section, we list and comment the expected conditions that our implementation of Mercury assumes to be held in order to operate properly. These assumptions do not limit the system, but highlight the situations that would need extra consideration and design effort to solve in a real-life implementation of the model.

Products variability

We will not deal with any type of product, but with products from a certain predefined set. Although in brick-and-mortar retailers an object sold could potentially take any form or weight, by restricting the item types available we aim to simplify the information processing from sensors.

The set of products available to operate properly will fulfill the condition that no two of the objects weigh the same, and their weights will always be lower than five kilograms and greater than one hundred grams. What is more, all items will fit in a single shelf stand.

The conditions above suffice to ensure that we can use competitive and inexpensive force transducers, while removing the requirement of applying computer vision over the objects in the shelves to be able to distinguish when weights cannot be used as a discriminator.

Therefore, the assumption presented here for products variability can be used to create a more simple environment for testing, while keeping a set up close to the real one. Recognizing items is indeed already one research problem on its own, and further study of this matter will benefit the development of smart stores.

Number of shelves and cameras

In the infrastructure we present here, we will only use one shelf with two stands available. The setup for the number of shelves and stands scales up easily, and one shelf suffices to ensure that we use a realistic environment for testing purposes. The main considerations of using more shelves are that the system will have to process and coordinate efficiently the different sensors, as well as the information generated by them. In this sense, the modular approach taken by Mercury simplifies the process of enlarging the establishment.

The same considerations also apply to the number of cameras, that would need to be increased to cover a greater area if many shelves are added to the store. In Section 2.2.1, we did dive deep into the problem of considering the minimum number of cameras required to cover the whole market area.

Cameras and shelves placement

We will not solve the issues described in Section 2.2.1 for camera placement to reconstruct the market convex polygon, or deal with people overlapping effectively. Instead, we will assume that Mercury is built in an squared establishment that can be completely covered by just one camera in any upper corner of the room, no significant occlusion will occur neither. To recreate Mercury on a larger establishment the camera placement problem should be revisited to ensure a reliable coverage of the total area of the shop. Keeping an array of cameras is also worth considering to simplify and make more fault-tolerant the process of tracking and detecting targets in the store [148].

Crowding and store capacity

It may occur that in a real scenario there would exist zones in the shop with poor visibility, whether the space cannot be fully covered by the video surveillance system, or the zone is too crowded. Therefore, in addition to the assumptions made in the previous section, we will also consider that the space available will not become saturated by a high number of customers. That situation could lead the service to lose reliability of the data provided by the tracking system. That is why the store capacity will be considered as the maximum number of people that allows the system to generate reliable data regarding the tracking and pose estimation techniques. In our case, due to the limited room available for testing, no more than four people are allowed to be shopping simultaneously.

Customer behavior

There are many different behaviors that could take place in a establishment that are not part of the main shopping behavior and that we will not take into consideration in this work.

- **Customer group**. Having an entity that represents a group of customers in a way that whatever item taken by any of its members is automatically added to a common virtual cart is advantageous and interesting (e.g. for families or friends). However, as it is not a fundamental feature, it is not going to be available in this project so that we will require every user to register individually.
- **Leaving products on the floor**. We will not cover the situation in which a customer leaves an item on the floor – or any other surface other than the shelves – and does not recover it later. We will assume that the customers still holds it, making the applicable bank charge when leaving the shop.
- **Exchange of customer products**. Despite products exchange between customers that are shopping together could be a real-case scenario, we will not make the correspondent update on their virtual carts. Thus, the customer that was firstly holding the product will be charged with its price.
- **Mislaid objects**. Another concerning events are the ones around product loss which we will not cover in this work. These situations may occur when a customer accidentally drops an object or when a product falls off the shelf due to a bad positioning. In the first case, the user would be charged with the proper price of the product and, in the second case, the product will be removed from the service.

Misleading conducts such as not registering at the shop entrance, laying or sitting on the floor, covering the video surveillance, adding external objects to the shelves, or similar inappropriate behaviors will not be contemplated.

Payment processing

As we focus on creating the infrastructure for a smart store, third-parties partnership for payment processing and validation is not a relevant part of the scope of this project. Therefore, we will not register any bank account on the customer profile nor consider real or fictional payments.

Moreover, it could happen that a customer is charged an erroneous bill leading them to file a complaint or claim. However, we will not study nor cover those administrative procedures.

Natural disasters and other unexpected events

There are common issues to other establishments such as power outages, internet connection going down or natural disasters that, for the sake of simplicity, will not be handled in this work.

4.1.2 Use cases

By taking into consideration the previous section, where we presented some simplifications made to the model, we can define the use cases that our implementation of Mercury will cover. These scenarios allow customers to fully experience the shopping process. Customers will be able to enter the shop, take whatever they want, and just walk out. Furthermore, customers can request clarifications on their last receipt.

On the following, we will explain how the infrastructure reacts to some use cases that are also presented as activity diagrams at the end of the subsection.

Customer enters the shop

The shopping process begins when a customer enters the smart store. In our case, the customer has to be registered at Mercury iOS App – that we will present at Section 4.4.2 – and has to click a specific button to start shopping. Once the button is tapped, the app will establish connection with the Client Manager server (Section 4.4.1), that will notify that a new customer has arrived to the Store Manager (Section 3.2.4). Finally, in case there was not any other customer shopping inside, the Store Manager will notify the Computer Vision Module to start its tracking service.

This process is captured in the diagram introduced at Figure 4.2.

Customer performs action on a product

Once a customer has entered the shop, they can start changing the state of the store by grabbing or leaving items from the shelf. In this section, we will deal with both events as a singular *action*, since the way the system response to each is analogous.

Our infrastructure for the Computer Vision Model of Mercury includes an intruder area close to the shelf so, when a customer enters that region, an alarm is triggered, and the module starts storing the pose estimation of the customer along with the tracking metadata.

Consequently, when a customer performs an action (grab/leave) on a product from a shelf, the sensors located at the shelf will notify that there was a change (resp. decrement/increment) on the shelf weight, and the Shelves Module will send the metadata along with the timestamp of the event to the Store Manager. The Store Manager will then request the pose estimation and tracking metadata, for the given timestamp, to the Computer Vision Module in order to create a job at Job Dispatcher module to assess which customer performed that action. After that, the

prediction will be sent to the Store Manager, which will notify the Client Manager in order to update the particular virtual cart of the customer at the mobile application.

This scenario, that involves every Mercury module, is graphically reproduced at Figure 4.3.

Customer exits the shop

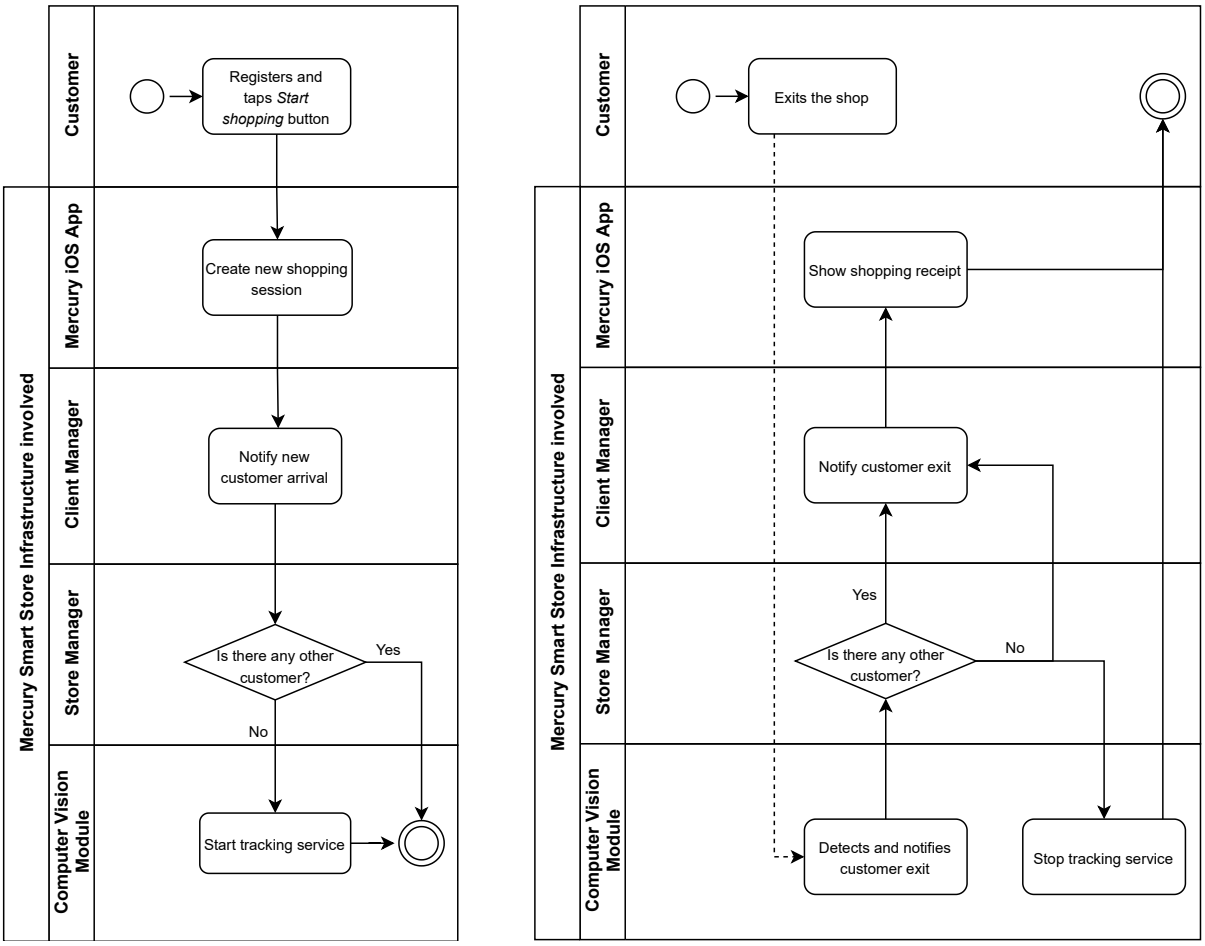
When a customer exits the establishment, the tracking service will detect that there is one person less. Hence, the Computer Vision module will notify the Store Manager. This component will send back a request to stop the tracking service, if that customer was the last one inside the shop. In any case, the Store Manager will notify the Client Manager that the customer has exited in order to propagate that information until it arrives at the mobile application, where the ticket with the price breakdown would be displayed.

This procedure can be seen at its corresponding activity diagram at Figure 4.2.

Customer requests an explanation on a product

Once the customer has exited the shop, the receipt with the price breakdown is immediately displayed. This receipt contains, for each product, the option of requesting an explanation for its purchase. Therefore, if the customer requests a clarification, the mobile app will notify it to the Client Manager, which will scale that request to the Store Manager.

Then, the Store Manager will retrieve metadata along with past predictions from the database and will send it to the Explanation Module, which will reproduce the scenario in order to generate a human-understandable explanation. That explanation will be sent to the Store Manager, that will pass that clarification through the Client Manager in order to display it at the customer's mobile. Finally, the whole process is shown at Figure 4.4.



((a)) Customer enters the shop ((b)) Customer exits the shop

Figure 4.2: Use Cases: entering and exiting the shop

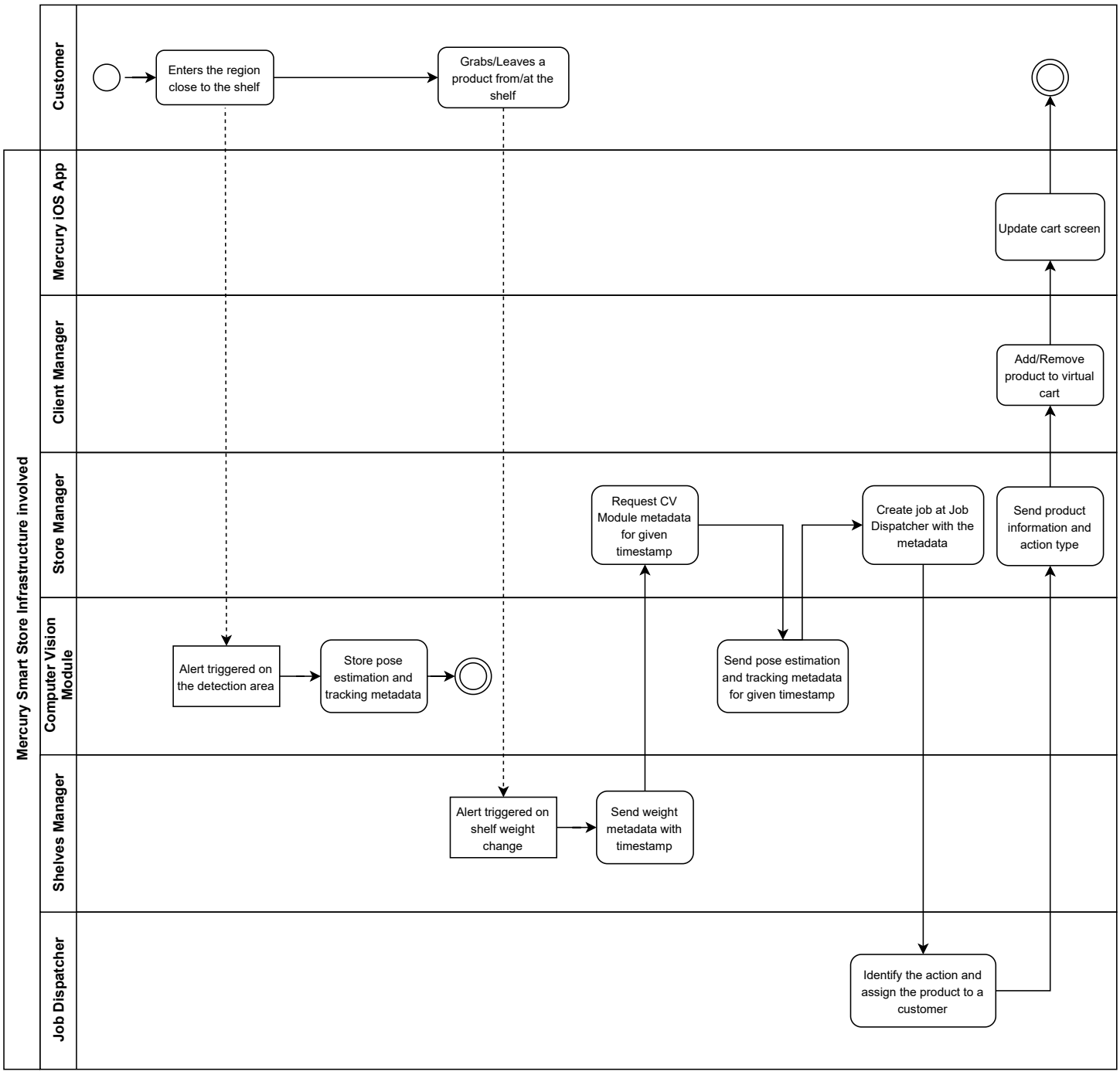


Figure 4.3: Use Case: Customer performs action on a product

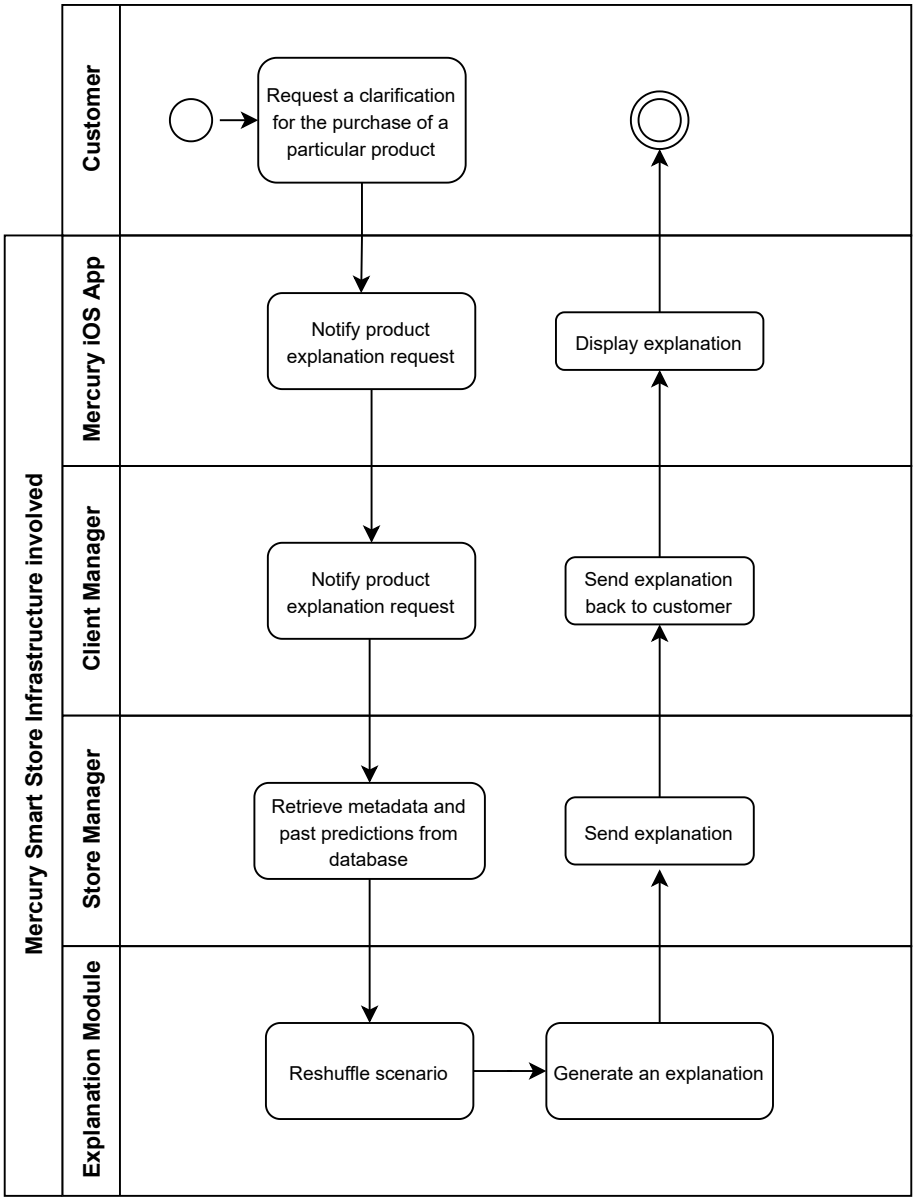


Figure 4.4: Use Case: Customer requests explanation on a product

4.1.3 Development and deployment

The development repositories for our implementation of the Mercury Smart Store model can be found on Github [149], a Git repository hosting service that fosters collaboration and communication between developers, and enables them to work together on different projects. The code is split up in different repositories under the Mercury organization[1].

Note that there are several concerns that must be taken into consideration when deploying our implementation of Mercury, or when trying to work on the development of these technologies. Although library dependencies and guidance on how to install each module is covered in the Github repositories readme files, we include in this section a table describing what programming languages were used to build each module, the integrated development environment software used, as well as the operating system platforms currently supported.

Module	Programming Language	Integrated Development Environment	Platforms supported
Shelves	C/C++	CLion	Windows
Computer Vision	Java, JavaScript	IntelliJ	Multiplatform
Client	Swift, JavaScript	Xcode, VS Code	iOS[2]
Store Manager	Java	IntelliJ	Multiplatform
Job Dispatcher	Java	IntelliJ	Multiplatform
Explainability	Python	PyCharm	Multiplatform

Table 4.1: Technical notes on the implementation proposal of the Mercury modules

4.2 Shelves Module

We devote this section to cover the inner working of the Shelves Module. Then, we move to briefly explain how the communication is handled with other modules, by introducing the metadata format generated by this component. Finally, we present the smart shelf we designed for our own implementation of this module.

4.2.1 The smart shelf

We created a smart shelf by attaching load cells along with electronic scale modules to a customized wood ledge. Then, we connected the sensors to a microcontroller which directly communicates with the Store Manager to report any update in the state of the shelves (see Fig. 4.5).

We decided to use load cells, as we required a transducer which could convert mechanical force into a measurable electrical output. To measure weights with these cells, we also needed to have a need-regulated power source, an output amplifier (noise reduced), and an ADC converter [150]. These requirements are met through the usage of the HX711 integrated circuit[3]. This circuit does not required pre-programming for the internal registers, as it can be completely controlled through the pins. The inner working of this component allows it to perform readout

[1]Link to the Mercury organization repositories in Github: `https://github.com/Mercury-Smartstores`

[2]Though the server that handles the communication with the mobile app is cross-platform, the mobile app was developed for iOS.

[3]HX711 Datasheet: `https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf`

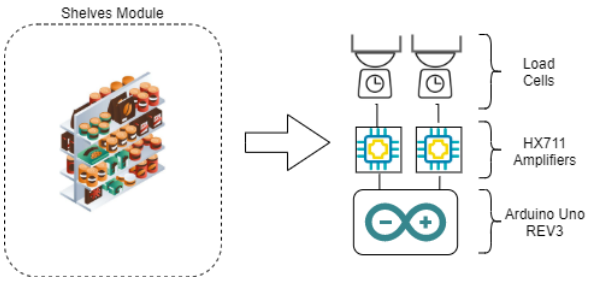


Figure 4.5: Mercury Smart Shelves implementation diagram detailed

from a Wheatstone bridge, formed by the load cells, converting the analogical results to digital with a 24 bits ADC converter. Figure 4.6 shows the load cells along with the HX711 amplifier setup.

As a mean of managing these components we decided to use the Arduino Uno Rev3 microcontroller[4] which is based on the ATmega328P board[5]. To develop the program for this microcontroller we decided to use the framework PlatformIO[6] along with the CLion[7] integrated development environment, they both provide a way of testing and releasing to production Arduino based software.

The program that reads from the HX711 module relies on the library developed by Bogdan Necula [151], and it was coded in *C++*. Before executing the program, the load cells dividers must be set to the correct values. These load cells divider correspond to the parameters that the amplifiers use to estimate weight as a measurement of the force applied to the cell. The following formula introduces this relationship:

$$Weight_{real} = \frac{Weight_{read}}{Cell\_Divider}$$

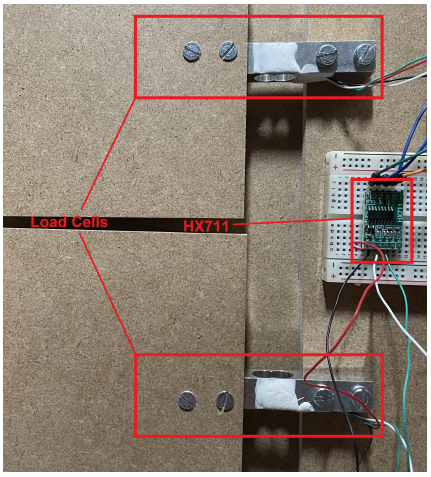


Figure 4.6: Load cells and HX711 configuration in a wood ledge for testing Mercury.

[4]You can find more information about this microcontroller on `https://store.arduino.cc/arduino-uno-rev3`.

[5]ATmega 328P Datasheet: `http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf`

[6]PlatformIO: `https://platformio.org/`

[7]CLion IDE: `https://www.jetbrains.com/clion/`

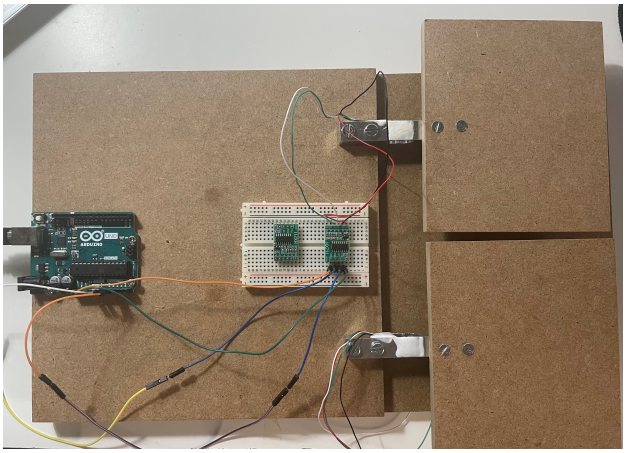


Figure 4.7: Mercury smart shelf.

In order to calculate the *Cell_Divider* for a given load cell, it suffices to measure an object which weight is known, and clear from the formula the value. It is also important to notice that the load cell will make several measurements to take the average, and report any value. This means that the impact of fluctuation is minimized.

All in all, Figure 4.7 shows the complete setup for the smart shelf.

4.2.2 Serial bus communication

The microcontroller is assumed to be directly connected to the same computer that runs the Store Manager. For this reason, direct readouts can be made from the serial port. Note that all Arduino boards contain at least one serial port (UART or USART), and some have several [152, 153].

As we deployed the Store Manager in a Windows computer, we made use of the Win32 API [154] to be able to create a file for the expected COM port, to which the Arduino is connected, so that we are able to handle the I/O communications. In Windows, by reading from the COMSTAT structure[8], the flag cbInQue, we know if we are capable of reading the bytes emitted by the serial Arduino printer.

The program which manages the communication is also directly connected to the Store Manager via a socket. The messages that the Store Manager and this system exchange are limited to weight measurement variations. In this sense, the Smart Shelves Modules notifies whenever a new object is placed on the shelf, or whenever an object is removed from the shelf. The metadata format generated by this module is shown below:

Listing 4.1: Metadata file format for change of store state triggered

```
[Integer]-[String]-[Float]-[Timestamp]
```

The value for the fields exposed before can be enumerated as it follows:

1) The integer value for the shelf stand number. It represents the stand impacted by the change, in our case there are just two stands available so this number can be just 1 or 2.

[8]More information: `https://docs.microsoft.com/en-us/windows/win32/api/winbase/ns-winbase-comstat`.

2) String value with the name of the action. There are only two feasible actions, *added* for new items placed in the shelf or *removed* for items taken.

3) A weight float measurement of the item removed or added.

4) Timestamp when the action took place.

This data format fulfills the requirements of easily communicating the information from the Shelves Module to the Store Manager. Note that the condition exposed in Section 4.1.1 on the weight of the products holds, so every object can be fully discriminated from the others just by using the data contained in the metadata file described above. The timestamp is required to be able to unify later the information obtained via different sensors, and reshuffle the scenario to decide who performed a certain action.

4.3 Computer Vision Module

The main objective of the Computer Vision Module is to gather information regarding the state of the store to be able to later infer what happened when the store state changes. For this reason, the module must be able to not only retrieve video or images from cameras or optical sensors, but also process the media content to produce metadata for the smart store system to consume.

Our implementation of the Computer Vision Module is split up in two components: the Camera Manager and the Pose Estimation and Tracking Service. The task of image and video managing is assigned to the Camera Manager, which completely manages the camera in the store. On the other hand, the Pose Estimation and Tracking Service are invoked to produce metadata whenever it is necessary. Figure 4.8 shows the detailed diagram for this module.

As we have stated in Section 2.1.1, Bosch smart cameras integrate Intelligent Video Analytics tools which can be used already for tasks such as tracking [31]. Nevertheless, there are several concerns that arise from the usage of this tracking method. The first one is that most of the Bosch devices are optimized to work with car or intruders datasets, as their devices are designed for surveillance tasks. For this reason, customer detection would not provide a great performance and accuracy results. Furthermore, the camera training program to apply the smart camera features requires an extra dataset to pre-train the model to be able to operate successfully in a different environment.

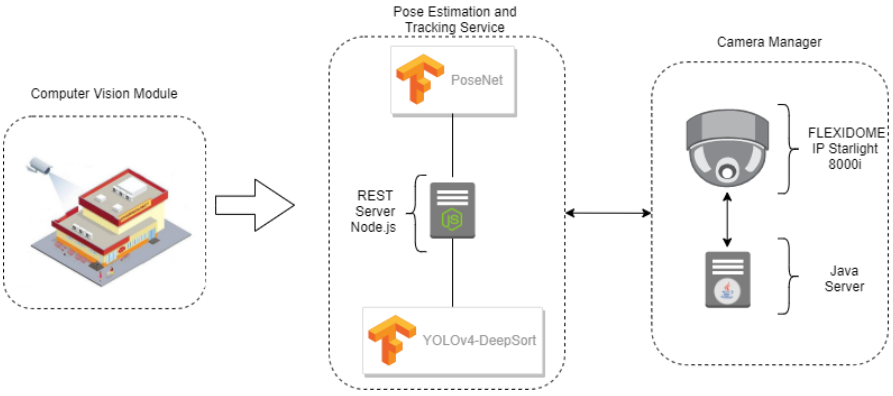


Figure 4.8: Mercury Computer Vision Module diagram detailed

As our main objective was not to focus on a tracking model, but to develop a new cashier-less store concept, the tracking functionalities were completely delegated to the Pose Estimation and

Tracking Service. We chose state-of-art and competitive tracking approaches for our case, which was people walking inside a building. Furthermore, as we opted to use an approach with an object detector, the service could be improved in the future to also keep track of the stock in the store, as other studies have already suggested [47, 155].

4.3.1 Pose Estimation and Tracking Service

The Pose Estimation and Tracking Service is in charge of generating appropriate metadata responses to requests made by the Camera Manager. To build this architecture, we decided to use a REST Server [156] in Node.js[9], since we only needed a stateless protocol with just a set of predefined feasible requests for the different services. The server was created using Express, being cross-platform so that it can run in any machine, even a different device from the Pose Estimation and Tracking Service (provided there exists a common network to connect both). This server will receive requests from the Camera Manager to start generating metadata. In particular, the following high level operations are available:

- **Estimate pose.** This request will be made along with a stream of images or a single image for which the service will have to estimate the poses.
- **Enable tracking.** This request will start the tracking mode, so it must be sent along with a new video streaming channel.
- **Disable tracking.** This request will shut down the tracking mode and close the opened channels.

In order to estimate the pose from a certain image, we make use of the PoseNet model [114] in its TensorFlow.js form[10]. The underlying network architecture of our implementation of PosNet is RestNet50 [115]. Secondly, to track customers inside the store we chose to work with the YOLOv4 [101] object detection model combined with the DeepSORT [106–108] algorithm.

The metadata generated for pose estimation requests is a JSON file with the timestamp value of the time when the request was made. The file also contains the score predicted value for each body part, the name of the part of the body that it references to and the X and Y coordinates where the part was predicted to be. You can find the format described and exposed below:

Listing 4.2: Metadata file format for Pose Estimation

```
[Timestamp]
{
    "frame": [Integer],
    "poses": [
        {
            "success": true,
            "result": [
                {
                    "score": [Float],
                    "part": [String],
                    "position": {
                        "x": [Integer],
                        "y": [Integer]
                    }
```

[9]Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. You can find more information on its website: `https://nodejs.org/`

[10]Github repository: `https://github.com/tensorflow/tfjs-models/tree/master/posenet`.

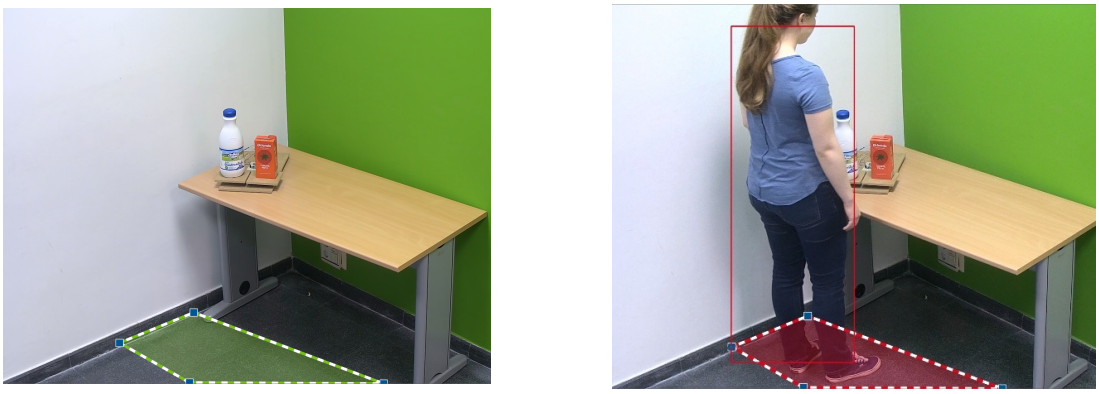
```
                },
                ...
               {
                    "score": [Float],
                    "part": [String],
                    "position": {
                        "x": [Integer],
                        "y": [Integer]
                    }
                }
        },
        ...
    ],
    "frame": [Integer],
    "poses": [...],
    ...
    "frame": [Integer],
    "poses": [...]
}
```

On the other hand, since for tracking we perform estimations continuously through a video streaming channel, the Pose Estimation and Tracking Service reports updates regularly for the frames that it is receiving. The first time that the tracking engine starts working, it records the beginning timestamp and it starts enumerating frames. For each frame, if any object is detected, then it provides a detailed explanation of the object reported with the respective estimated properties: the object id (each object is assigned an ID so that we can track objects detected in consecutive frames with the same ID), object class (e.g. person, milk bottle, ...)[11], box coordinates (the points which delimit the box that can be drawn to fully identify the object). Lastly, when the tracker is shut down, an end timestamp is sent. A general example of the metadata format for the tracking service is depicted above:

Listing 4.3: Metadata file format for Tracking and Object Detection

```
Begin: [Timestamp]
Frame #:  [Integer]
<Tracker ID: [Integer], Class: [String],
BBox Coords (xmin, ymin, xmax, ymax): ([Integer],
[Integer], Integer, [Integer])>
...
<Tracker ID: [Integer], Class: [String],
BBox Coords (xmin, ymin, xmax, ymax): ([Integer],
[Integer], [Integer], [Integer])>
Frame #:  [Integer]
<Tracker ID: [Integer], Class: [String],
BBox Coords (xmin, ymin, xmax, ymax): ([Integer],
[Integer], Integer, [Integer])>
...
<Tracker ID: [Integer], Class: [String],
BBox Coords (xmin, ymin, xmax, ymax): ([Integer],
[Integer], [Integer], [Integer])>
End: [Timestamp]
```

[11]Note that we currently only track customers, so the implementation of Mercury restricted the estimations to only the object class *person*.



((a)) Example of region

((b)) Customer entering the region

Figure 4.9: Mercury intrusion region close to shelf

4.3.2 Camera Manager

The Camera Manager component is made up of the smart camera FLEXIDOME IP Starlight 8000i and a Java server which handles the communication with the camera, the Pose Estimation and Tracking Service, and other external modules from Mercury.

We decided to configure the FLEXIDOME IP Starlight 8000i to create an intrusion region to detect customers entering the area close to the shelves. To program the Bosch smart camera we used the Configuration Manager[12] they provide to their customers. This decision was taken to ensure that we were not computing pose estimation in every moment, but just when customers are close to the shelves. In this sense, we aimed to reduce the computational costs of running the Mercury store model. Figure 4.9 shows the region set up to trigger an alarm when a moving object enters.

The alarm set up with the Configuration Manager can later be read by the Java server. The Java server not only manages the communication with the camera using the ONVIF procotol [157], but it also directly connects with the Node.js server set up for the Pose Estimation and Tracking service, and it handles the communication with other external Mercury modules such as the Store Manager.

4.4 Client Module

Earning customer trust is an important objective for Mercury. Hence, enabling real-time communication and interaction with them by developing a infrastructure with a front-end side seems to be a good way to increase customer engagement [158].

As the majority of the adult population of advanced economies (and several emerging economies) has a high rate of smartphone ownership [159], delegating the communication with the customer to their smartphones seems to be an accurate manner to bring this technology closer to potential customers. In the following subsection, we will propose a feasible implementation to achieve this objective.

The task of dealing with the *human-side* of the service is achieved by introducing two submodules. The Client Module division can be seen at Figure 4.10.

[12]Configuration Manager manual: `https://resources-boschsecurity-cdn.azureedge.net/public/documents/Configuration_Manage_Operation_Manual_enUS_9007200472721035.pdf`.

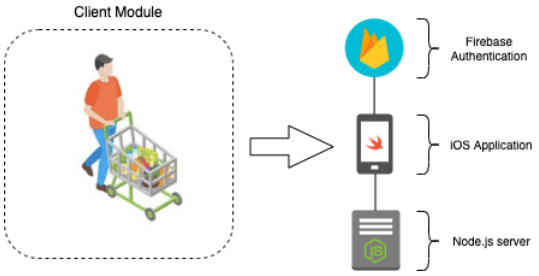


Figure 4.10: Mercury Client Module detailed diagram

On the one hand, it contains a submodule – named *Client Manager* – that consists of a Node.js server that enables real-time, bidirectional and event-based communication, and works on every platform, browser or device. This server will act as intermediary between the Store Manager Module (see Section 3.2.4) and the front-end side of the whole infrastructure (i.e. the mobile application).

On the other hand, we built a mobile application that will improve the shopping experience. Users will need to register in the application to be able to enter the shop. Once inside the shop, they will be able to see the updates in the shopping cart when they grab or leave a product from the shelf. After leaving the store, they will see a pop-up with the ticket containing the price breakdown for the products they took, next to a video camera button, in case they want to require an explanation on the addition of that specific item to their cart. Besides, they can always check their past receipts on the app.

4.4.1 Client Manager

As introduced before, the Client Manager aims to facilitate the communication between the customer and the overall store operation. To achieve that, we firstly created a Express[13] application, which is a minimal and flexible Node.js web application framework, to provide a quick and easy way to deploy a robust API and a simple routing for requests made by clients.

Secondly, due to the ease of integrating Swift with the client side of Socket.IO[14] framework – we are able to write code that runs natively on iOS while maintaining the simplicity and expressiveness of a JavaScript client –, we decided to deploy the server also with Socket.IO.

Besides, we added handling for different message types that the server will send to the client application according to the information received from the Store Manager. These messages are:

- Add a product to the virtual cart.
- Remove a product from the virtual cart.
- Customer is exiting the store.

On the other hand, the client side will be able to send these two messages to the server:

- Customer is entering the shop.
- Customer has requested an explanation on a product.

[13]Express framework: `https://expressjs.com/`
[14]Socket.IO: `https://socket.io`

The message format for both cases could be shown as:

Listing 4.4: Client-Server messages exchange format

```
"message": "message ID",
"data": [
    "custom-key": custom-value,
    ...
]
```

Then, we just need to deploy the server as a usual Node.js application and start exchanging information with the client by listening at a certain port and host.

Finally, it is worth to mention that we used the Visual Studio Code[15] IDE since it has support for JavaScript language as well as Node.js debugging, although to run the Node.js application, it is required to install the Node.js runtime on the server.

4.4.2 Mobile application

For a better visualization for the customer, and in order to enhance their experience in Mercury, we built an iOS application called *Mercury iOS App* that we present in this section, as well as its features and introduce the screens layout. To yield this, we used Swift[16] as the programming language and Xcode[17] IDE due to is the integrated development environment designed by Apple for its operating systems.

Mercury iOS App is the realization of the front-end side of the system, that allows real-time visualization of the shopping status of the customer and, in consequence, their interaction with the store. Customers will be required to create an account, if they do not have already enrolled, in the app before entering the shop. That authentication process is handled thanks to the project created in Google Firebase[18] (Figure 4.11 shows the Authentication screen at Firebase console), that has a free service that manages all real-time data in the database so the data exchange is easy and quick [160].

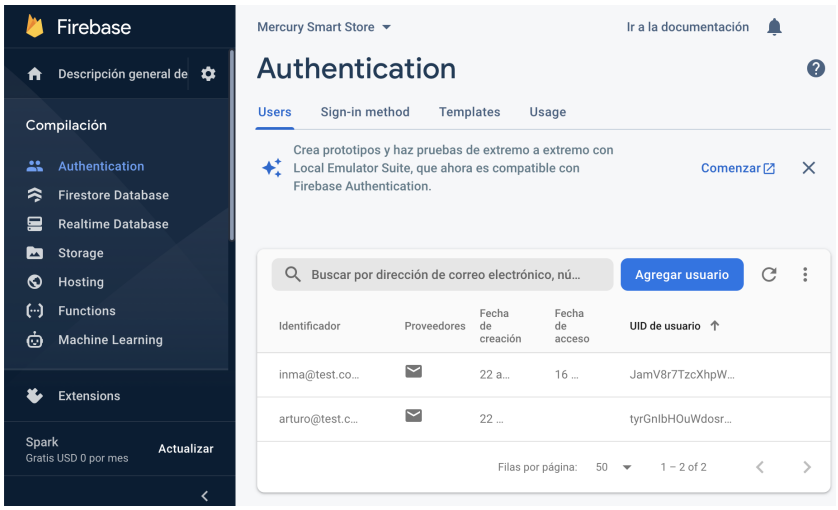


Figure 4.11: Mercury Smart Store project's authentication screen at Firebase console

[15]Visual Studio Code: `https://code.visualstudio.com/`
[16]Swift: `https://developer.apple.com/swift/`
[17]Xcode: `https://developer.apple.com/xcode/`
[18]Firebase: https://firebase.google.com/

Therefore, at the beginning, customers can proceed to authentication in the screens that Figures 4.12(a) and 4.12(b) present.

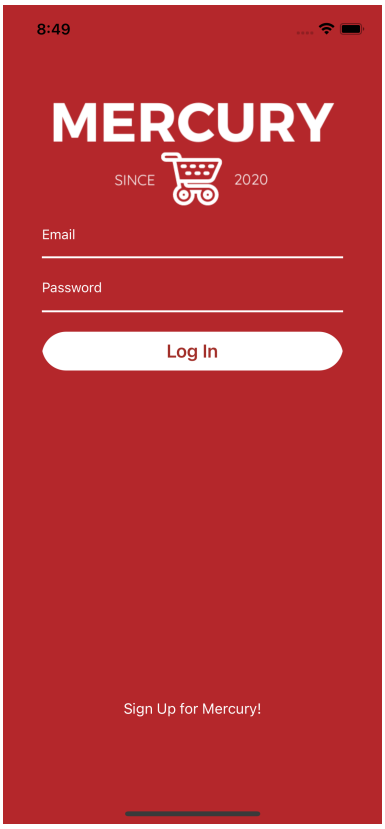
Inside the app, they will be able to navigate through three different screens: *Home*, *Cart* and *Tickets*. Hence, once the customers are logged in, they will be redirected to the home screen (see Figure 4.12(c)), from where they can log out from their account as well as notify the server their intention to enter the shop by tapping the *Start shopping* button. Meanwhile, the cart screen will display the message shown at Figure 4.12(d).

Once the customer has tapped the *Start shopping* button, they will be connected to the server – by using the Swift client of Socket.IO[19] framework – and moved to the cart screen, that will display an empty cart. Furthermore, the home screen will change it status and disable the button interaction while user is inside the shop, returning back to the enabled status when the customer exits the establishment. These two features can be seen at Figures 4.13(a) and 4.13(b), respectively.

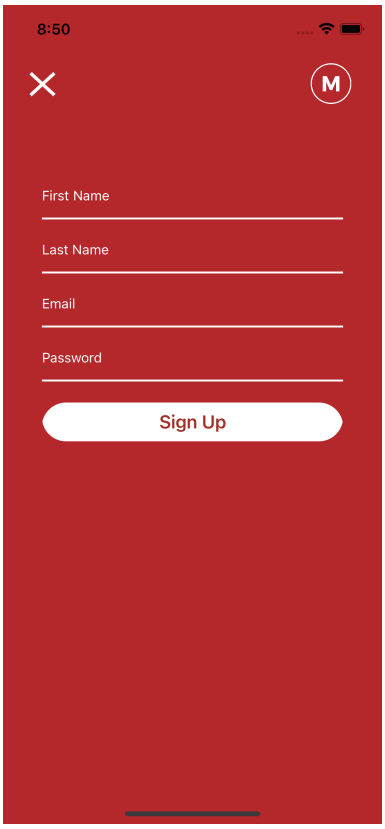
Then, they can freely move inside the shop, behaving according the simplifications of the model (see Section 4.1.1). As they modify the store status by grabbing or leaving products from the shelves, their virtual cart will be also updated in real-time. When they are done and walk outside the shop, the Store Manager will notify the event to the Client Manager server, that will send a message to the app so that the ticket is displayed to the customer. In case the customer did not buy anything, a message will be displayed as Figure 4.14 shows. On the contrary, they will be able to see the receipt containing the items taken, the price breakdown and a video-camera button in case they want to request an explanation on a particular product. If they request a explanation, they will be presented a tagged video in which they will be able to see the specific moment in which they took the particular item.

Finally, if the customer wants to check previous purchases, the *Tickets* screen will collect every past receipt, allowing they to see the ticket detail with the price breakdown as well (see Figures 4.13(c) and 4.13(d)).

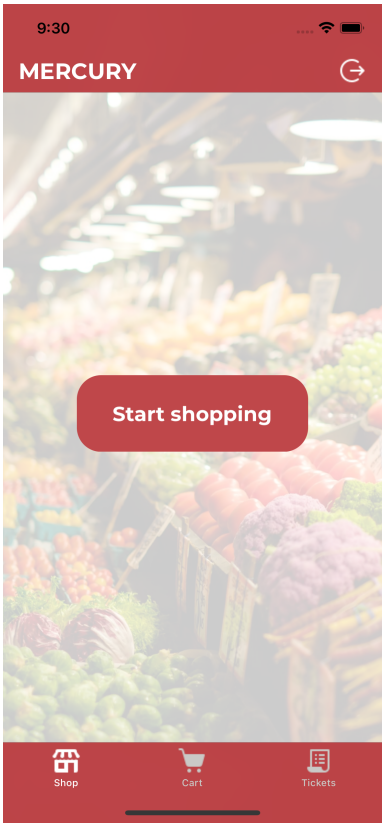
[19]Swift client for Socket.IO: `https://socket.io/blog/socket-io-on-ios/`



((a)) Mercury iOS App: Login screen



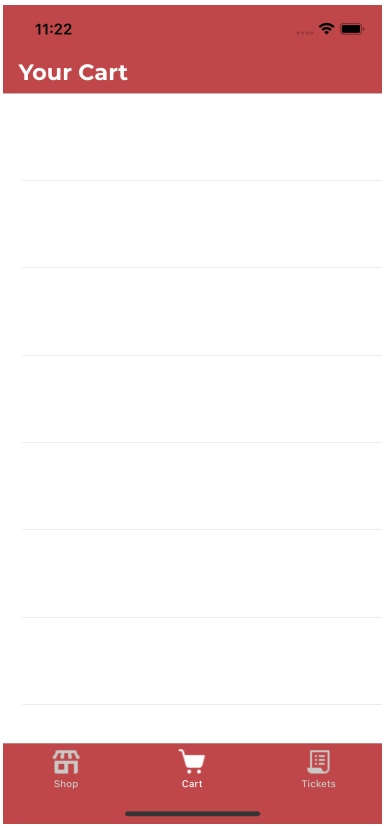
((b)) Mercury iOS App: Signup screen



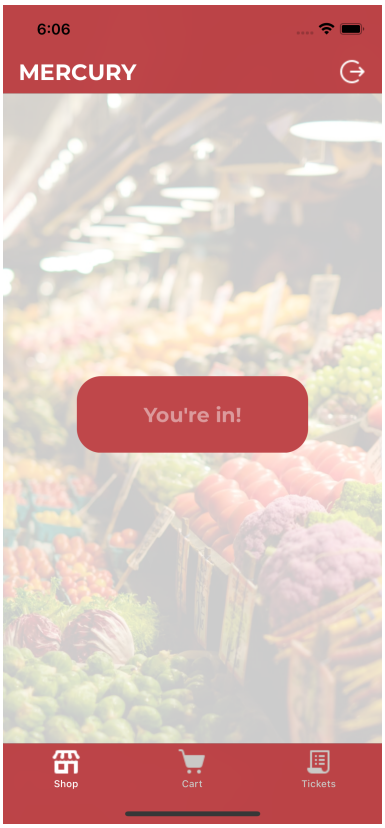
((c)) Mercury iOS App: Home screen when customer is not the shop

((d)) Mercury iOS App: Cart screen when customer is not the shop

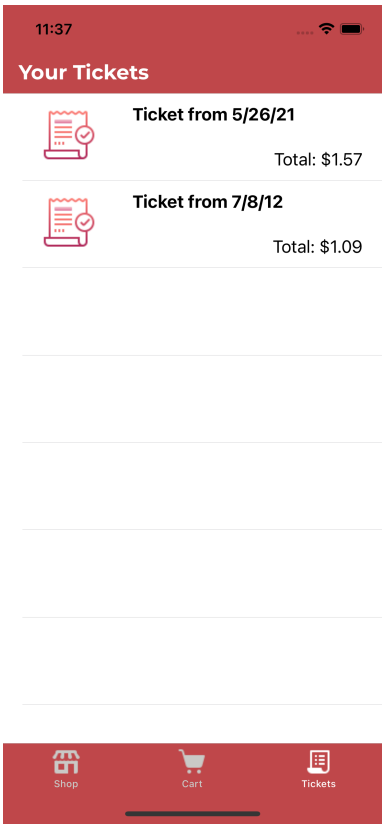
Figure 4.12: Mercury iOS App authentication screens and screens after login



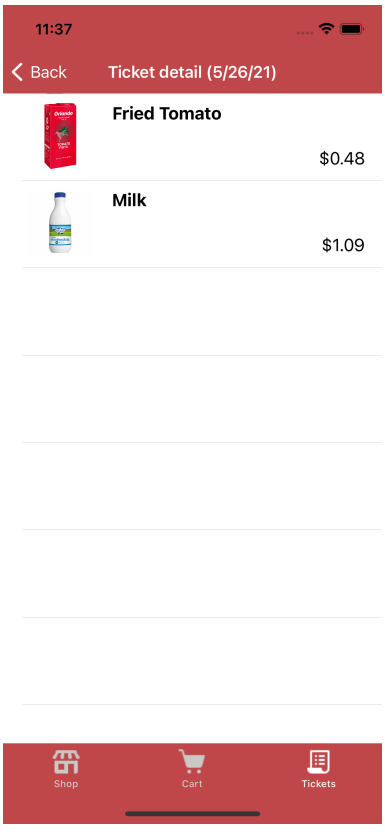
((a)) Mercury iOS App: Cart screen when customer has just entered shop



((b)) Mercury iOS App: Home screen while customer is in the shop

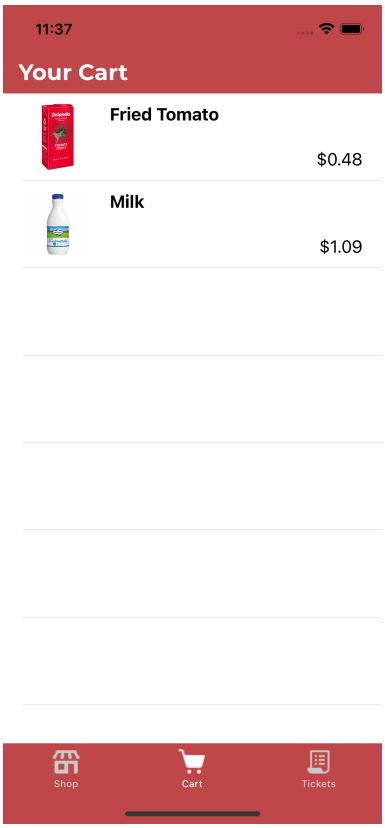


((c)) Mercury iOS App: Past tickets screen

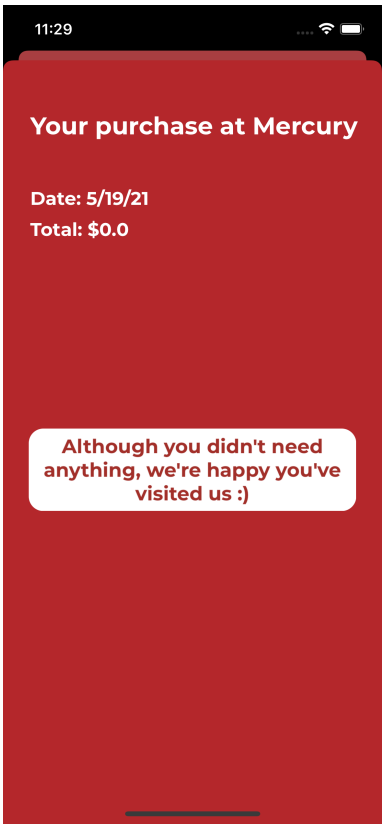


((d)) Mercury iOS App: Past ticket detail screen

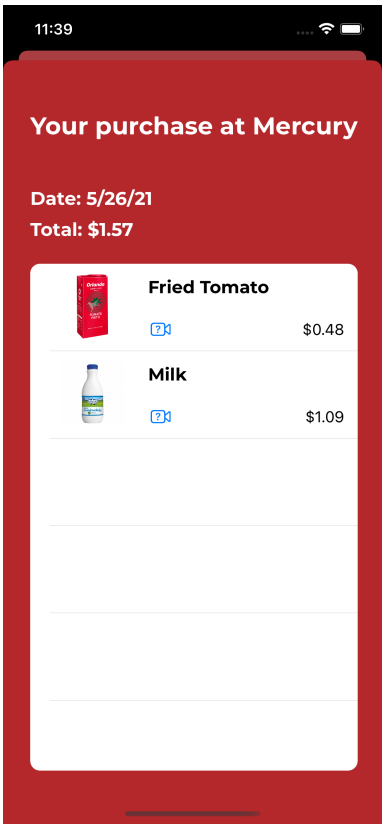
Figure 4.13: Mercury iOS App screens when the customer enters the shop and screens for past tickets



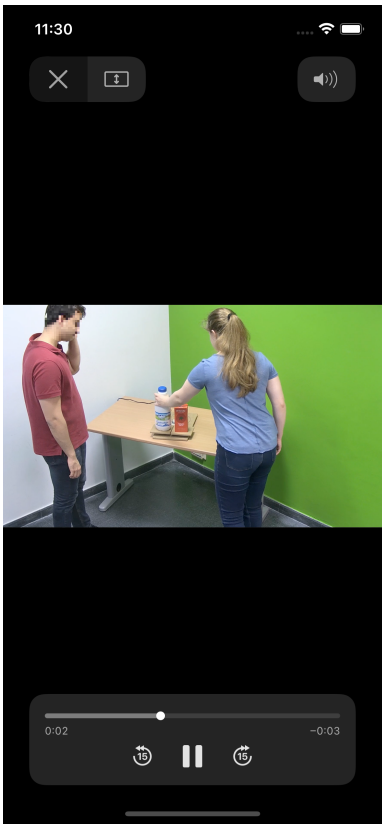
((a)) Mercury iOS App: Cart screen when customer has grabbed some items



((b)) Mercury iOS App: Ticket popup displayed when customer exits the shop with no purchase



((c)) Mercury iOS App: Ticket popup displayed when customer exits the shop



((d)) Mercury iOS App: Explanation by displaying a tagged video

Figure 4.14: Mercury iOS App screens for shopping status

4.5 Store Manager

The main goal of the Store Manager is to govern the Mercury infrastructure. In this sense, this component acts as a communicator between different modules, and it can perform requests on demand to solve issues. For example, if a customer is charged with an item, the Store Manager can request the Client Module to send a notification accordingly.

For the Store Manager we came up with an implementation based on a multi-client Java server, that handles the communication with all modules as well as the access to the database. Figure 4.15 details the implementation of this module.

We did not focus on the implementation of the database managing access, since we wanted the module to be completely database-agnostic we relied on common database managing operations that would be easy to formulate in any paradigm.

Figure 4.15: Mercury Store Manager Module diagram detailed

On the other hand, the inner working of the Java server is pretty straight-forward. The server starts running and creates a thread to connect via a socket with the different modules. Then, depending on the information received from any of the interfaces it decides the next step to perform. The following table exposes all of the different events that could occur, highlighting the module that triggers the event and the recipient of the action, as well as the next high level actions that would be required to apply:

Sender	Event	Recipient	Next action
Client Module	Client enters the store	Computer Vision Module	Start in-store tracking (if not active)
Shelves Module	Item removed/added to a shelf	Computer Vision Module	Request metadata with timestamp close to the action
Computer Vision Module	Send metadata for a time-span requested	Job Dispatcher	Send metadata with timestamp in the range of the event
Job Dispatcher	Return a job result	—	Stores the results
		Client Module	Send a notification to the customer
Client Module	Request a clarification	Explainability Module	Send metadata and job result to reproduce the scenario
Explainability Module	Send a clarification	Client Module	Redirect the clarification
Computer Vision Module	Notify a customer left the store	—	Store the information

Table 4.2: Events and next action schedule for the Store Manager

All in all, our implementation of the Store Manager achieves a well-orchestrated infrastructure

by relying some of the tasks to the different modules. For instance, the Computer Vision Module is only invoked to provide metadata when an action occur, so the component itself is in charge of managing the idle metadata produced.

4.6 Job Dispatcher

To build the Job Dispatcher, we decided to use a Java server that would work concurrently for all requests and asynchronously with respect to the Store Manager. With this setup, we wanted to focus on the task of correctly estimating all job instances at the cost of delaying the response and updates to the customers virtual carts. The Java server triggers a new thread execution for each new incoming job, the thread finishes once a job result, assessing feasible candidates for the action, is generated. Figure 4.16 shows the diagram of the implementation.

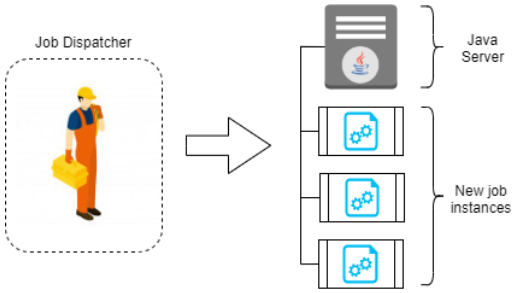


Figure 4.16: Mercury Job Dispatcher diagram detailed

There are different aspects worth revisiting before introducing the inner working of this component. For instance, we should review the metadata information available:

- Human pose estimated for the customers around the shelves.
- Position of customers around the shelves and their bounding boxes.
- The positions of the shelves[20].
- The type of the products.

With this information available the easiest approach is to compute the distances between the shelves and the hands of the customers. However, before performing this task we had to take into account that though the position of the customers around the shelves is known for every timestamp, the correspondence of customers to human poses is not known. For this reason, we include a subsection to discuss about the strategy followed to assign human poses to customers for which we know the position. After that, we move to introducing the algorithm used to assess the probability of a customer being the one who performed the action. Finally, we briefly detail the format of the job result sent back to the Store Manager.

4.6.1 Linking human poses to customers

There are many alternative approaches for assigning human poses to customers, from a simplified application of the DeepSORT [106, 107] to many other alternatives tracking alike exposed in Section 2.2. Nevertheless, we can take advantage of the knowledge of the bounding boxes for the customers to swiftly compute the identity.

[20]Note that this information is known before-hand since it is fixed for a certain store.

Figure 4.17: Example of customer pose inside a box

We know that all of the human body parts for a certain person are structured in a skeletal metadata, so we can iterate through all the keypoints and count the number of points that lie inside the bounding box of each customers. After that, we can simply pair the customers with the poses that has more human keypoints inside its bounding box. Figure 4.17 shows the skeletal drawn for a customer along with the bounding box provided by the tracker system.

The algorithm to link human poses to customers can be described as it follows:

Algorithm 1: Linking human poses to customers ids

```
poses = getHumanPoses();
boundingBoxes = getBoundingBoxes();
mapIdToPoses = createMapPosesToId();
foreach pose ∈ poses do
    bestScore = -1, bestId = 0;
    foreach box ∈ boundingBoxes do
        score = box.countKeypointsInside(pose.getKeypoints());
        if score > bestScore then
            bestScore = score;
            bestId = box.getId();
    end
    mapIdToPoses[bestId] = pose;
end
```

As we know that bounding boxes are rectangular, it is quite simple to estimate if a point in the plane lies inside the shape. Nevertheless, since we also have some quadrilateral shapes for the smart shelves represented in the 2-dimensional image, we implemented a general algorithm to assess if a point is or not inside a convex quadrilateral. In the appendix A we expose some computational geometry approaches used in the implementation of some of algorithms for the Job Dispatcher.

4.6.2 Computing best candidates

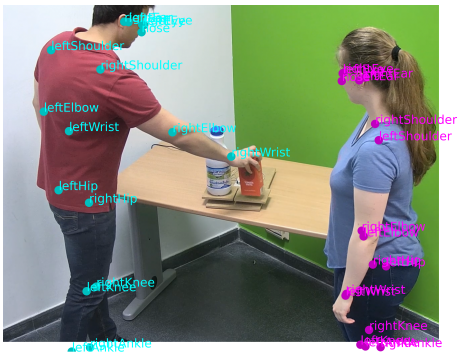
Once we have the metadata tagged to match each customer, we can proceed to evaluate the customers. On our implementation, we maintain maps that assign to each time frame a new map that contains the relationship of customer identities with their metadata. In this sense, we have two maps, one for the metadata regarding the pose and another for the bounding box. Furthermore, we restrict the time frame analyzed just to the frames capturing the object lifting or dropping. By doing this we guarantee that the access to the information is not computational intensive and we focus on the data that provides the most interesting insights to the action that occur. Also, we assume that we know before-hand the height of each product, so that given a

product of a certain type and a shelf containing the product, we can build a rectangular prism that fully contains the item.

The general idea of the algorithm is to compute the minimum distance between the reconstructed prism, which contains the item and the quadrangular shelf, and the wrists of each customer that has entered the intruder region. Then, we compare that distance with the distance to the intersection of the diagonals of the bounding boxes of the customer to get the lowest value. This ensures that, even when partial occlusion occurs to hide the analyzed human body parts, we can calculate the metric. For these cases, we just assume the distance to be infinity. Moreover, then we can formally define a probability assigned to each of the customers depending on the minimum distance calculated.

((a)) Original image

((b)) Shelves prisms



((c)) Pose estimation over customers

((d)) Pose estimation wrists

((e)) Distance from wrists to shelves

Figure 4.18: Analysis of the metadata information to compute best candidate

Mathematically, we can express the algorithm above as it follows. We denote w_i^R and w_i^L to the right and left wrists, respectively, for a certain i customers. Also, at the cost of slightly abusing terminology, we name *dist* to the function that takes objects or human body parts and output

its distance. Finally, we denote b_i as the bounding box assigned to the i customer and S_{item} as the shelf prism which depends on the item on the shelf. Then, we can calculate what is the fittest customer according to the strategy described before for a set of n customers:

$$D = \min_{i \in \{1,\dots,n\}} \left(\min \left(dist(w_i^R, S_{item}), dist(w_i^L, S_{item}), dist(b_i, S_{item}) \right) \right)$$

Indeed, taking advantage of the notation introduced before we can introduce the computation to assign a probability to the customers depending on their closeness to the shelf. Lets denote P as the probability function we want to compute, thereby:

$$P(i) = \frac{D+1}{\min \left(dist(w_i^R, S_{item}), dist(w_i^L, S_{item}), dist(b_i, S_{item}) \right) + 1}$$

To prove that $P(i)$ is well-defined it suffices noticing that

$$\min \left(dist(w_i^R, S_{item}), dist(w_i^L, S_{item}), dist(b_i, S_{item}) \right) \geq 0$$

for all i and therefore $D \geq 0$. Hence, $\min \left(dist(w_i^R, S_{item}), dist(w_i^L, S_{item}), dist(b_i, S_{item}) \right) + 1 \geq 1$ and $D + 1 \geq 1$, so neither the numerator or denominator nullifies. Besides, it is always the case that $D \leq \min \left(dist(w_i^R, S_{item}), dist(w_i^L, S_{item}), dist(b_i, S_{item}) \right)$ and therefore the ratio given by $P(i)$ varies from $(0, 1]$.

Note that the cases where $P(i) = P(j)$ when $i \neq j$ are the ones that Mercury cannot discriminate. In this sense, this formulation fully captures and highlights the limitations of the model. As a toy model, Mercury aims to reduce this cases but they can occur. Whenever this is the case, we simply do not assign the item to any customer.

All in all, the algorithm for the strategy that has been introduced is exposed below:

Algorithm 2: Computing best candidate in a Smart Store

```
prism = shelf.createPrism(item);
poses = getHumanPoses();
boundingBoxes = getBoundingBoxes();
candidates = getSetCandidatesId();
bestCandidate = -1, bestDistance = +∞;
foreach candidate ∈ candidates do
    pose = poses.getPose(candidate.id);
    box = boundingBoxes.getBox(candidate.id);
    leftWristDistance, rightWristDistance = +∞, +∞ ;
    if pose.isDefinedLeftWrist() and box.isInside(pose.leftWrist) then
        leftWristDistance = distance(pose.leftWrist, prism);
    if pose.isDefinedRightWrist() and box.isInside(pose.rightWrist) then
        rightWristDistance = distance(pose.rightWrist, prism);
    scoreWrists = min( leftWristDistance, rightWristDistance);
    candidateDistance = min( scoreWrists , distance(box, prism) ));
    if candidateDistance < bestDistance then
        bestCandidate = candidate.id;
        bestDistance = candidateDistance ;
end
```

The idea underlying the algorithm is also captured in Figures 4.18.

4.6.3 Job result file

When the Job Dispatcher is over assessing all the customers next to the shelves, using the algorithm described before, then it produces an output file named the job result file. This file contains the following information:

- The number of customers around the shelves when the event took place.
- The distance from each customer close to the shelf prism, following the guidelines proposed in the previous subsection.
- The number of customers who scored the highest likelihood.
- The probability value computed for each customer.
- A string indicating the body part used to compute the distance (or the bounding box in case of no visible wrists) for the best fitted customer. In case there are more than one, a concatenation of values separated by a comma is provided.
- The product type and the stand number.

4.7 Explainability Module

To work with explanations we decided to use a Python server connected to a database with information from past cases, mainly to be able to apply a Case-based Reasoning strategy [161]. The diagram that captures this implementation is shown in Figure 4.19.

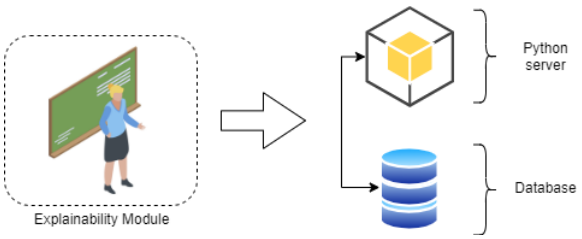


Figure 4.19: Mercury Explainability Module implementation diagram detailed

We split this section into two, one for each explanation type depending on the class of the user making the request: customers or store managers. For customers, we developed a system that provides visual explanations, whereas, for the staff, we provide explanations based on past cases annotated.

4.7.1 Customers

Whenever a customer request a clarification on why an item was charged to their account, we proceed by reproducing the scenario with the metadata received. This is, we recreate the scene when the object was taken, and modify the visualization to include metadata that explains the interactions of the customer with the store. In this sense, we proceed as it follows:

1) We identify the customer making the request and blur all other customers in the video scene using the information from pose estimation and tracking to obtain their positions.

2) We crop and zoom the video to focus on the area around the shelf from which the customer took the object, since we know, from the information reported by the smart shelf, which stand was impacted.

3) We trim the video to include just the exact moment when the product was taken. For that purpose, we take advantage of the given timestamp.

4) We send the video modified with extra information such as the timestamp, the product name and the amount charged.

4.7.2 Staff members and store managers

Staff members and store managers require information that explains the behavior of the system rather than clarifying an action. In this sense, the explanation that must be delivered differs from the previous one. We now focus on explaining why the system behaved the way it did, so that any error or bug can be noticed and fixed appropriately.

To implement the strategy to provide the explanations, we followed the classical 4R-cycle (Figure 4.20), that can be described as follows [147]:

1) **Retrieve** the case(s) from the database that are more similar to the new problem at hand.

2) **Reuse** the solution(s) from the retrieved cases to propose a new set of feasible solution options for the new problem.

3) **Revise** the proposed solution to assess the suitability according to the context.

4) **Retain** the new case/problem and its revised solution so that in can be use in the future when it is appropiate.

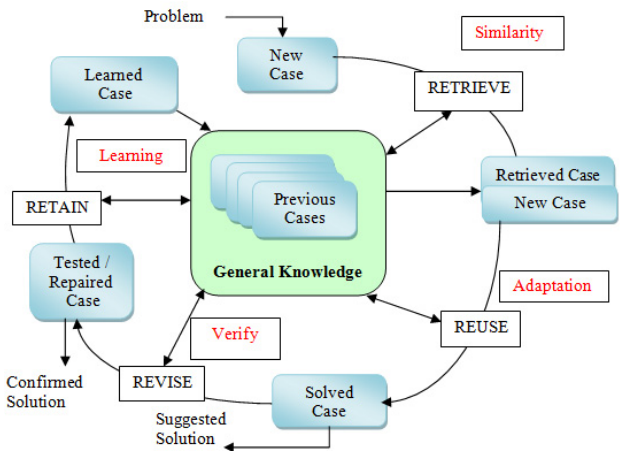


Figure 4.20: CBR classical 4R-cycle[21]

We now briefly describe how each of these stages is applied in our particular case. Firstly, we retrieve the new case from the Store Manager, that is the one in charge of sending the issue request to the Explainability Module. This request contains metadata from the information gathered from the sensors in the scene, as well as the resolution from the Job Scheduler packed in the job result. Then, we start comparing information such as number of customers involved,

[21]Source: ResearchGate https://www.researchgate.net/figure/Case-Base-reasoning-R4-cycle-3-The-knowledge-support-system-framework-for-ACS-In-this_fig1_277935871

number of customers that scored with a probability of 1 (or number of customers that scored with a probability more than 0.9), type of distance reported by the highest score (for example, if the highest score was achieved by measuring the distance from the left or right wrists or the bounding box), the position of the customer relative to the shelf, etc. In this sense, we can denote each case C_i as a vector of size equal to the number of attributes we are comparing. Conversely, we name C_k to a generic past case.

To compare cases, we use the likelihood function defined below, which emulates the one already introduced by Montazemi and Gupta [162]:

$$\mathcal{L}(C_i, C_k) = \frac{\sum\limits_{j \in A_i \cap A_j} w_j^k \cdot sim(C_i^j, C_k^j)}{\sum\limits_{j \in A_i \cap A_j} w_j^k} \tag{4.1}$$

where C_i is the new case and C_k a past case we are comparing against, and A_i and A_j are their respective set of attributes. The values w_j^k are called *weights* of the attribute j for the k past case, and they are a measure of the importance of the attribute. Also, we name each attribute $\alpha \in A_i$ of a C_i vector case as C_i^α. Additionally, the function $sim(C_i^j, C_k^j)$ compares the attribute j instance for the new case C_i and the past case C_k. In order to compute the function $sim(C_i^j, C_k^j)$, we propose a strategy which considers the maximum value M_j that any attribute j can take, and then we can express the function as it follows:

$$sim(C_i^j, C_k^j) = 1 - \frac{|C_i^j - C_k^j|}{M_j} \tag{4.2}$$

As for the weights w_j^k, we considered them as contextual-importance values which help discriminating among previous cases. To quantify these values, we compute the variation of the standard deviation σ_j of the attribute j for the d previous candidates analyzed:

$$\sigma_j = \sqrt{\frac{1}{d}\sum_{k=1}^{d}(C_k^j - \bar{\mu}_j)^2} \quad \text{where} \quad \bar{\mu}_j = \frac{1}{d}\sum_{k=1}^{d} C_k^j \tag{4.3}$$

Hence, the importance of the j attribute to discriminate cases, with respect to a new case C_i, can be seem as the relative importance of the attribute:

$$w_j^k = \frac{\sigma_j}{\sum\limits_{p \in A_i \cap A_k} \sigma_p} \tag{4.4}$$

After assessing the new case C_i against past cases, we must decide which case fits better the new instance. For example, if it is the case that there are a great number of customers and the explanation provided for a past case was that occlusion occur because of crowded scene, we can infer that there is probably a problem too with the occupancy. In order to achieve this, we take the best similar case according to the output reported by the likelihood function $\mathcal{L}$. However, there are two impasses that may occur: the situation is unknown because there is no memory available to match the new case, or there are several cases that apply with the same degree of confidence. In both situations, we assume an alarm is triggered so that an expert intervenes to select the best fit among all the options, or to provide a new different explanation. Once we

are done revising the case, we might require a new explanation provided by an expert. If a new explanation is introduced in the system, we tagged the case and store it in the database so that if in the future a similar issue arrives we know how to solve it.

Regarding the solution (i.e. the explanation provided), we decided to include a textual clarification along with a video modified with metadata tags. The video is always generated and it does not depend on the case, and it is mainly a recreation of the scene recorded but adding the human poses estimated, the bounding boxes with the identities of the customers, and the distances used by the Job Dispatcher to assess the candidates taking or dropping an item. With respect to the textual clarification, the expert can define its own template to match new similar cases. The template contains text information describing the scene as well as variables that are updated for each particular case. The Figure 4.21 shows a template example for a crowded scenario in which the system cannot predict which customer took a certain product.

ATTRIBUTES	
Description	*Value*
Number of customers close to shelf	n
Likelihood of highest value customers	s_1
Number of customers scoring the highest likelihood	m_1
Shelf impacted by the action	s
Object (taken or dropped)	o
Action (take or drop)	a
EXPLANATION	
Action a performed on shelf s, containing o. When the event occurred, there were n customers around the shelf. There were m_1 customers scoring the same highest likelihood (s_1). For this reason, the system was prevented from properly assessing who performed the action by occlusion in a crowded environment.	
Similarity Score	α

Figure 4.21: Template for occlusion issues

The attributes available for any template are the following ones:

- The number of customers close to the shelf, n.
- The likelihood of customers who scored the highest value, s_1.
- The likelihood of customers who scored the second highest value, s_2.
- The pose estimated score prediction for the customer who scored the highest likelihood[22], p_1.
- The pose estimated score prediction for the customer who scored the second highest likelihood[22], p_2.
- The number of customers who scored the highest likelihood of performing the action, m_1.
- The number of customers who scored the second highest likelihood of performing the action, m_2.
- The distance from the highest scoring customers to the shelf, d_1.

[22]In case there is more than one customer with the same probability, we consider the mean value of their pose estimated score predictions.

- The distance from the second highest scoring customers to the shelf, d_2.
- The shelf impacted by the action, s.
- The object taken or dropped, o.
- The description of the action, whether it was an object taken or dropped, a.

4.8 Conclusions

We have introduced an attainable realization of the Mercury smart store model by detailing how each and every component is built. Besides, we also highlighted scenarios that are not covered by our implementation.

In the first place, we opted to implement the Smart Shelves unit by using a smart shelf we crafted for this project. This device does not only include a weight sensor, but it can also communication with the Store Manager.

Regarding the Computer Vision Module we explained how we applied the DeepSORT and YOLOv4 models to build up a tracking service. Moreover, we also introduced system we used with the PoseNet model to estimate the pose of the customers in the store. Finally, we introduced the IP Camera that was used to record and detect customers next to the shelves.

On the other hand, for the Client Module we developed a mobile application as well as a server to handle the communication with customers. In this regard, we described how this component handles the notifications sent to the mobile application, completely owning the relationship with clients. We also detailed the iOS app we built for testing the system. Conversely, for the Store Manager that manages the communication among all the devices we specified how it was built using a multi-client Java server.

With respect to the Job Dispatcher, we introduced the algorithmic approach to assess customers, as well strategies to link human poses to customers using other metadata available. Ultimately, we also introduced and explained the resulting file that is output by the unit when it finishes analysing an event.

At last, we presented the Explainability Module by establishing how explanations are generated for each customer type.

All modules considered, our implementation of Mercury can be conceived as an end-to-end realization of a cashier-free smart store.

Chapter 5

Scope, suitability and scenarios

In the previous chapter, we introduced Mercury as a straightforward model of a cashier-free smart store system and detailed its different components. Since the technology that is used in the modules is complex and there is a myriad of possible scenarios that may occur in a shopping environment, our implementation of Mercury operates under some specific premises. In this work, we have assumed that the existing products in the shop belong to a predefined set, where no two objects weigh the same. Moreover, those products can only be placed in the available shelf that contains two stands.

Besides, the infrastructure works under the assumption of little occlusion so there will not exist poor visibility in the shelf area. Thus, the system will not allow customers to cover the shelf or misbehave in any other way such as leaving products on the floor or exchanging products with other customers. Nevertheless, the detailed list of simplifications can be seen at Section 4.1.1.

All in all, in this chapter we will present both cover and uncovered scenarios by the realization proposed in the previous chapter as well as suggest ways to make Mercury implementations become more resilient.

5.1 Covered scenarios

In this section, we will show different scenarios in which our implementation proposal works as expected. Since the complexity of the resolution depends on the number of customers that are performing actions on the store state, in the following we detail how the system behaves when dealing with particular single and multiple customers scenarios.

5.1.1 Single customer scenarios

In case there is a single customer in the shop (as shown in Figure 5.1), the model resolves every event with little effort since it has only one candidate to assign as the source of the shopping state changes.

Therefore, when an item is taken or left at any stand of the shelf, the performance of the system is the one detailed at 4.3 but simplified, as the Job Dispatcher can immediately identify the customer that has performed the action.

It is worth to mention that, as this assignation is trivial, the infrastructure can also operate in cases with more occlusion and every possible event will be successfully covered.

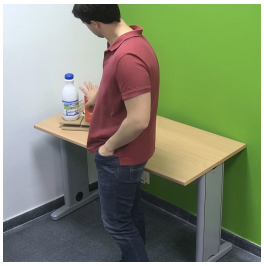


Figure 5.1: Single customer scenario

5.1.2 Multiple customer scenarios

When more customers enter the scene, the task becomes harder. Thanks to the conditional activation of the Computer Vision depending on the occupancy of the intruder region defined in areas close to the shelf, we can differentiate the following two sub-scenarios.

One single customer close to shelves

This case is almost analogous to the one in which a person is the only customer in the shop. As we know the intruder area is the critical zone from where customers can perform actions on the shelf. If there is just one customer inside it, the assignation problem, in spite of being a multiple customer scenario, becomes once again trivial, since the Job Dispatcher will ignore the rest of the customers.

Multiple customers close to shelves

In previous scenarios we have taken advantage of having the intruder region to reduce the complexity and the calculations of the tasks that the Job Dispatcher resolves. In this case, we have a tougher task since there is more than one customer inside that area and, therefore, close to the shelf and maybe willing to perform an action. We distinguish two sub-cases that we present as follows.

One single customer performing an action
The system also successfully covers the case in which two or more customers are located close to the shelf (see Figure 5.2). In this situation, the Job Dispatcher distinguish the source of the change in the shop state thanks to the metadata of the pose estimation and tracking service along with the sensor data. It will point out the shelf that has been involved in the change and calculate the probability (see Algorithm 2) for each candidate to have performed the action.

Figure 5.2: Multiple customers close to the shelf; single customer performing action

Multiple customers performing non-crossing actions
This scenario is the most critical one since more than one customer is performing an action.

((a)) Non-crossing actions ((b)) Crossing actions

Figure 5.3: Multiple customers performing actions

Under the store context in which Mercury operates, there are two possible options to occur (see Figure 5.3): customers take their closest item or customers take the furthest one.

Our implementation of Mercury achieved the resolution of the non-crossing situation by taking advantage of the assumption that we know in advance the products set so we are aware of the height of each element; therefore, we can build a rectangular prism that fully contains the item. Hence, for each item, the Job Dispatcher module will calculate the minimum distance between the prism that contains the product and the quadrangular shelf and the wrists of each candidate (given by the pose estimation metadata), and compare it with the distance to the intersection of the diagonals of the bounding boxes (provided by the tracking metadata) to get the lowest value.

However, this calculation does not hold if the customers are crossing their arms, not only because of the occlusion produced in certain moments but also because in the analyzed video frames we noticed that the metadata provided by the different modules is not enough to calculate reliable distances to resolve the conflict.

5.2 Uncovered conflicting scenarios

Although, as we have stated, there are several scenarios in which our proposal for Mercury operates properly and even solves moderately complex situations, there are others where the system behavior is not well-defined, or it simply does not have enough data to correctly assess customers performing actions. For this reason, in this section we cover all the conflicting and uncertain scenarios for which special attention must be paid.

5.2.1 Crowded stores and occlusion next to the shelves

Clearly the system will be drastically compromised if heavy occlusion occurs next to the shelves or inside the store. For instance, the Computer Vision Module will not be able to detect or track properly the customers inside the store. Besides, if there are obstacles blinding the shelves region it will not be possible to apply any strategy to assess customers when the state of the store changes. In this sense, though the Mercury model is not limited to a single camera, our proposed toy model was. For this reason, occlusion has a particularly great impact on our setup.

However, even if a resilient camera network is built to offer a full coverage of the empty establishment, since the customers themselves represent moving obstacles, it is not possible to guarantee that blind spots are not going to appear. This implies that the Mercury proposed model is theoretically limited by the capacity of the Computer Vision Module to operate rightly.

5.2.2 Multiple customers touching the same product

Another conflicting scenario emerges whenever two or more customers are directly touching an item but only one takes it. This problem does directly impact our implementation of Mercury, since we computed a probability of being a fitted candidate based on the distance to the object. When multiple customers share a common distance it is not possible to distinguished who performed the action among all of them.

Nevertheless, in the general specifications of Mercury the system is not limited to using a measurement as the one proposed on our implementation. In this case, it would be possible and reasonable to think of another methods to come up with a way of computing scores for customer candidates. The next section will cover this topic.

5.2.3 Weights variability

Although we explicitly exposed in the simplifications that we were going to consider the weight as a complete discriminator for the set of items in the store, the assumption itself is not realistic for a supermarket and many other retail establishments. Note that many groceries are delivered using standard quantities and units that are sometimes shared by different types of products.

In this sense, though our implementation is directly impacted by this scenario, the general Mercury model is not. It is still possible to implement Mercury by using a different strategy to compute which item is placed on each shelf. We will discuss about this in the following section.

5.3 Production refining of the model

In this section we propose some improvements and extensions worth considering for any realization of the Mercury smart store model.

5.3.1 Working in crowded environments

As we introduced in the previous section, crowded scenarios represent a problematic situation for the Computer Vision module. One way to address this issue is to replicate the cameras in different positions to ensure that the system is more robust while we minimize the blind spots.

Unfortunately, as we stated, in an scenario of moving obstacles it is not possible to ensure a complete coverage of the scene in a crowded environment. Consequently, it seems reasonable to set up a limit capacity for the establishment depending on the room available to ensure that a good camera coverage is guaranteed (and, hence, the Computer Vision module can operate properly).

Ultimately, it is worth highlighting that the problem of crowded environment does not only impact smart stores but brick-and-mortar establishments too. When many customers are grouped in a certain small area, it is not possible to avoid having issues such as theft offenses.

Multi-camera environment

The Mercury model is easy to extend due to its modular approach. Since the Computer Vision component is a self-contained unit, we can duplicate it or change its inner working to include more cameras. The only changes that would be required are within the communication protocol of

the Computer Vision module with the Store Manager, to include the new metadata information available from the module. The increment in the number of cameras could drastically improve the performance of the system in crowded environments, as we have previously stated. Besides, it would increase the coverage of the room.

Furthermore, a greater camera network would potentially enhance the sensor fusion network not just in a redundant way, but also making it a complementary and cooperative network. Having more cameras would mean that we could implement 3D reconstruction to compute three dimensional distances in the store, increasing the reliability of the system while reducing the impact of occlusion (check Section 2.2.2). Indeed, 3D reconstruction has already being implemented by some smart stores such as the one introduced by Standard Cognition [111].

The main scalability difficulties are then related to the reconstruction of the three dimensional scene, or the proper placement of the cameras to ensure a great coverage. Note that both problems have already been deeply studied by computer vision researchers (see Section 2.2).

5.3.2 Product recognition using computer vision

One great feature that is worth considering in any implementation of Mercury is product recognition. This feature can be conceived as a double edged sword since it requires staff members to keep updating their systems so that they are always learning how to recognize new products. Unfortunately, as we have already mentioned, products in grocery and retail establishments tend to vary drastically even for short periods of time (seasonal packaging or new package designs for marketing purposes). On the other hand, they can be of great help when assessing the customer who is most likely of performing an action. Besides, it can get rid of the problem of forcing products to be of different weights to ensure we can identify them.

Product recognition could be implemented in the Computer Vision module to ensure that, in situations when several customers report the same probability of being the one performing an action (e.g two or more clients simultaneously touching an item or equally close), we can effectively come up with the best candidate. In order to this, a strategy such as temporarily assigning to all customers equally probable the item and later analysing the video recorded to track the item can be used.

Finally, recognizing products can substantially simplify the problem of discriminating by weight the items. In this sense, the smart shelf could rely on any optical device to detect products or directly receive the information from the Computer Vision module.

5.4 Conclusions

In this chapter we have covered the situations that are managed rightly by our implementation of Mercury, as well as its limitations. Besides, we highlighted not just the limitations of our realization proposal but also some issues that might emerge when trying to implement the Mercury model in general.

Finally, we include some considerations worth reviewing to deploy the Mercury model in a real-world environment, as well as technology improvements that could pave the way towards a more robust system.

Chapter 6

Conclusions

Cashier-less shopping systems can help to improve the shopping experience but, as we have detailed in this document, pose notable design challenges, and sometimes require significant store redesign. As not much is publicly known about the technology behind cashier-free retail establishments, this work attempted to shed some light on the topic, and not only compile the state of the art of smart stores but also propose, detail and analyze a feasible solution. In this final chapter, we will recapitulate the results obtained during the realization of Mercury, and compile the advantages that smart stores bring to other environments, as well as some improvements to make Mercury implementations become more resilient.

6.1 Results

When we thought about writing this document, we asked ourselves which was the best approach so people with little context on the field were able to understand this work, know every possibility that has been attempted, and have a starting point to keep on with their own research and development of a smart store. With that in mind, we defined a list of objectives to achieve, that we will check in this section.

6.1.1 Objectives review

Taking a look back at the three main objectives defined in the introduction of this document at Section 1.2.1, we now proceed to review their accomplishment.

Objective [O1]: Identify technologies and strategies

First of all, we aimed to identify the main technologies and strategies used in the industry. We can see that during this document we have studied the ideas that are nowadays in the spotlight when creating a smart retail establishment, and we have presented examples of the materialization of those in the real life.

This information was compiled in the Chapter 2, where we went through different technologies applied at smart stores, such as smart cameras (e.g. IP Cameras), RFID tags, biometric identification (e.g. palm or face recognition), smart shelves, sensor fusion, and some computer vision techniques commonly used – like pose estimation or object tracking – when resolving the events. Moreover, we introduced strategies like recommender systems, and a feasible use case of computer vision explainability in the smart store environment.

Besides, we presented a wide range examples of smart stores that are operating in real life scenarios, and outlined the technologies that they officially confirmed to be using, or that researches suggested they may be using.

Objective [O2]: Propose a feasible cashier-free shopping model

During the Chapter 3, we suggested an architecture for a smart store model that we named Mercury, along with a set of objectives for each particular module that should meet every realization of the model. Furthermore, we incorporated some technologies and techniques proposed to address them. To yield that, we took into consideration the algorithms for sensor fusion and computer vision studied in Chapter 2.

The implementation of Mercury that we built was introduced in Chapter 4. In our proposal, we brought forward a set of smart devices and sensors to monitor the actions of customers inside the store. Besides, we designed and implemented an algorithmic solution to track customers and estimate their poses, as well as to distinguish who performed a particular action that modified the state of the store. Since it also seemed to be interesting to provide the system with the ability to explain its decisions, we added an Explainability Module. Apart from that, we proposed a system to handle real-time communication with customers that allows to show updates on their virtual carts, and provides clarifications for possible explanation they may request. In addition, we suggested a way for the different modules to communicate between them when we introduced the Store Manager Module.

Finally, we analyzed the cover and uncovered scenarios, as well as edge cases, of our Mercury implementation at Chapter 5.

Objective [O3]: Improvements to operate in a production environment

In the previous chapter, we exposed some improvements to ensure a proper working of the system in a real-world setup. Thanks to these improvements, we would effectively cover complex and conflicting scenarios, as well as provide a more fault-tolerant automated service.

6.2 Future work

Smart stores technologies are remarkable not only because of their achievements, but also because of the usage of techniques that are currently under the hood. In this section, we will discuss some improvements or features that can be considered when designing a smart store model. Likewise, we will introduce the extension of the model to other domains.

6.2.1 Other smart stores technologies

In Chapter 2, we reviewed some technologies that should be considered when designing a smart commerce. Then, in Chapter 3, we proposed an architecture that combines of some of them. However, in this subsection, we would like to highlight different technologies that have not been considered when implementing Mercury.

Bluetooth positioning

Implementations of tracking systems have become prevalent issues in modern technology due to its advantage of location detection of objects. In Section 4.3.1, we introduced a possible way to achieve so by combining YOLOv4 with DeepSORT algorithm, but, of course, that is not the only way to succeed in this task.

Today we see Bluetooth being integrated on a variety of devices [163]. While on some of these the only technology available for position tasks is Bluetooth, on others a cooperation between other technologies and Bluetooth would give an advantage in position tasks [164]. All in all, both of these methods can be used with dynamic positions so that Bluetooth technology has become another way to perform indoor object tracking. However, it usually requires line-of-sight operations, limited coverage and low-level programming language for accessing Bluetooth signal strength [165].

RFID and NFC tags

Near-field communication, such as RFID or NFC tags, is a feature that, as we have revealed in this document, is commonly used in the industry. There are numerous potential benefits of using them for all stakeholders in retailing industries. Some of these advantages include reducing labor costs, simplifying business processes, improving inventory control, increasing sales, and reducing shrinkage. Thus, with these technologies, a retail business can provide better customer service along with improvements in store layout. That is why many retailers and suppliers have already initiated various projects to utilize the technology, such as Walmart with RFID.

However, it has its limitations. The range for which we can use magnetic induction approximates to $c/2\pi f$, where c is a constant (the speed of light) and f is the frequency. Therefore, as the frequency of operation increases, the distance over which near-field coupling can operate decreases, and more resources would be required. Fortunately, inexpensive radio receivers have been developed with improved sensitivity so they can now detect signals, for a reasonable cost and power levels [166].

Recommender engines

Recommender engines can now be found in many modern systems that expose the user to a huge collection of products [167]. As we reviewed in Section 2.1.6, such engines typically provide the user with a list of recommended items they might prefer, or predict how much they would rather each item. These recommendations can often be found in online stores rather than the physical ones, since it is easier to create an online customer profile.

Moreover, we introduced a store that is currently dealing with some kinds of recommendations based on what the customer has already added to the cart (see Caper in Section 2.3.2). In addition to what Caper is doing, smart stores that include a computer vision module similar to one proposed in Mercury (see 3.2.2), have the advantage that they can be aware of the time that each customer spends on a certain area of the shop. This information can be really useful when developing a recommendation system that suggest the customer new products to add to the cart, based on their pleasures and most-frequently shelves visited.

Finance manager

Thanks to introducing a Client Module in the architecture (see 3.2.3), we can not only add recommendations as a new feature but also a finance manager.

As the purchase history of every customer can be both stored and displayed to customers, the system could include an expense manager that tracks the amount of money spent in each type of food (e.g. junk food, vegan products...) as well as include limitations on the purchase of those if the customer wishes. Furthermore, the customer will be aware of their monthly expenses.

6.2.2 Extensions to other domains

Since multiple technologies operate in Mercury, we can easily find an extension to other domains. For instance, the object tracking service could be utilized for surveillance systems to measure the occupancy of a place, which is useful for any kind of indoor space such as a museum, airport, university, or outdoor spaces like music festivals. Furthermore, combined with the pose estimation service along with the inclusion of intruder regions, or load cells – presented at the Computer Vision Module and Shelves Module (resp. sections 3.2.2 and 3.2.1) –, it could serve as a good theft or misbehavior prevention service (e.g. security in museums, palaces, etc) since the system could trigger alarms to the security staff based on the closeness of the visitors to the objects of interest (e.g. artworks).

Besides, those decisions taken by the system could be explained to the staff members by the Explainability Module (Section 3.2.6), as well as a list of events log or a video displaying the tagged action that triggered the alarm for a better understanding of the situation.

Finally, the myriad of metadata that is produced could be used to generate recommendations for the customers. In addition to the Client Module (Section 3.2.3), that proposes a effective way to increase customer engagement, techniques could be applied to customize customer journey by displaying those recommendations based on the sections or places that have visited inside a store, museum, etc. or products that they have grabbed or dropped in shops.

6.3 Final remarks

Mercury has given us the opportunity to dive deep in a wide range of fields by designing and developing hardware and software solutions. To yield that, we learnt about cutting-edge technologies that are in the spotlight when designing a smart store due to the variety of scientific publications we investigated and collated when doing our research.

Hence, we have learnt about ways to effectively communicate information between the physical elements of a smart store by designing the Camera Manager and the Shelves Module, and also to incorporate smart devices from which we can retrieve data than can be processed to make predictions.

Furthermore, we introduced ourselves to some AI fields such as Explainable Artificial Intelligence, when designing the Explainability Module, or Deep Learning, when looking for algorithms to design the pose estimation and tracking service. Likewise, we explored new horizons when looking for a catchy way to make Mercury attractive to customers by developing the Client Module.

Finally, this work has not only served us to learn about feasible manners of bringing to reality a smart store, but also to become competent when designing a vast architecture in which different programming languages, frameworks, platforms and physical devices are involved.

Appendices

Appendix A

Computational geometry approaches

In this appendix we introduce some concepts in Pasch geometry which can be of interest when implementing the Job Dispatcher in a Mercury smart store. We mainly focus on the plane geometry, which is the geometry naturally derived for the single camera implementation of Mercury.

A.1 Pasch Geometry

The axiom of Pasch is a statement for the plane geometry that was implicitly introduced by Euclid, and it cannot be derived from its postulates. The axiom was named after the mathematician Moritz Pasch, who played a key role on its discovery [168]. Without loss of generality, the statement mentioned can be characterized as it follows:

If a line, not passing through any vertex of a triangle, meets one side of the triangle then it meets another side.

Most of the results that are widely known for the plane geometry take for granted this axiom implicitly. For this reason, we will work in this section always under the assumption that the Euclidean and the Pasch principles hold.

Quadrilateral formation

In a 2D view of a store, customers can be fully contained in a quadrilateral figure. The same also applies to products recognized in the shelves or any other object. Besides, we can build quadrilaterals for each side of the prism built for the stands formed by the wood ledgers to encapsulate the items.

In consideration of the foregoing, given a set of 4 random points in the plane, we are interested in building up a quadrilateral (four-sided) figure. In particular, we will focus on what is known as convex quadrilaterals, this means that the four-sided figure will not have any interior angle greater than π radians. Some well-known convex quadrilaterals are the square, the rectangle, the rhombus and the trapezium.

A simple approach to the problem of building the convex quadrilateral from a set of 4 points is to use the Jarvis march algorithm [169]. This computational geometry algorithm is used to compute the convex hull of a given set of points in the plane, with a time complexity $O(nh)$ for n points and h points in the convex hull. After computing the convex hull figure, it suffices to check whether or not all the points are vertex of the figure.

On the other hand, the Hungarian mathematician Esther Klein observed that given any set of 5 points such that no 3 of them are collinear, then it contains 4 points that are vertices of a convex quadrilateral. The proof and the main idea of this result can be used to derived another approach for the quadrilateral formation problem. Indeed, this is the strategy implemented in our realization of Mercury proposed in this work.

Considering the approach made by Klein, in order to have a convex quadrilateral, the 4 points must suffice that no 3 of them are collinear. Moreover, if we denote the points as $p_1, p_2, p_3, p_4 \in \mathbb{R}^2$, then any of the segments $p_i p_j$ and $p_k p_l$ must intersect where $i, j, k, l \in \{1, 2, 3, 4\}$ and they are all different values. This algorithm presents a better complexity constant time $O(1)$, regardless of the points.

Point-In Polygon problem

In our realization of Mercury, we decided to use a probabilistic algorithm to link human poses to customers detected, by counting the total key points lying inside the box identifying the client. In order to this, we needed to solve the Point-in Polygon (PIP) problem, which has been deeply studied in the computational geometry field.

There are some great strategies already proposed to solve the PIP problem. One idea is to consider p the new point as a vertex of a triangle formed by other two points v_i and v_j which were two vertices of the quadrilateral. Then, we must only make sure that the sum of areas of all the triangles formed by taking four set of two vertices with $i \neq j$ and compare it with the area of the quadrilateral. If both areas are equal then the point is inside the polygon, otherwise it was not. Figure A.1 shows the differences in areas when building the triangles.

A similar approach can be conceived by comparing the angles from the vectors $\overrightarrow{pv_i}$, where $i \in \{1, 2, 3, 4\}$ denotes the $i-$vertex of the quadrilateral. In this sense, if we denote $\theta_{i,j}$ to the angle formed by the vectors $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$, then it can be proved that if p lies inside the quadrilateral then the following always holds:

$$\theta_{1,2} + \theta_{2,3} + \theta_{3,4} + \theta_{4,1} = 2\pi$$

Indeed, the previous strategy is the one implemented in the realization of the Job Dispatcher proposed in this document.

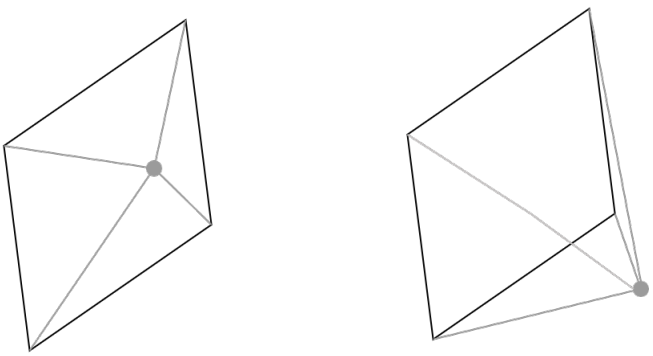


Figure A.1: Solution to the PIP problem for quadrilaterals using sum of areas

Appendix B

Contributions

In this appendix we detail the main individual contributions to the project. Due to the great extension of this project, we limit our exposition to just the most important contributions. If any task or part of the project is not explicitly mentioned, it is assumed to have been carried out by both members of the group together.

Some examples of tasks that we both worked on are the proposal of the objectives and key tasks for this project, the overall design of the document, the general Mercury model and implementation designs, the choice of the technologies used in our smart store, the testing of the different scenarios mocking the different Mercury modules, the unification of the metadata format used among all of the modules to communicate with each other, and the research regarding RFID technologies.

B.1 Contributions of Arturo Acuaviva Huertos

Due to the broad topic we were trying to cover in this project, we split up the revision of the scientific literature and public information available. Firstly, I mainly focused on studying the computer vision tasks with the only exception of explainability. In this sense, I reviewed the most important algorithms for object detection, multitarget tracking and pose estimation. Moreover, I had to study the different geometric alternatives proposed to deal with 3D image reconstruction, as well as to find out different ways to solve the camera placement and the correspondence problems. During my review of these topics, I encountered the problem of occlusion detection and handling, which we did not consider at the beginning of this project. Although we do not address this concern in our proposal of Mercury, I studied this matter to include a brief subsection in our state of the art. All in all, I had to not only find out the state of the art algorithms used to solve problems such as indoor tracking or product recognition, but I also studied their applicability, limitations, and how they worked.

Relating to the previous consideration, I also studied the main technologies used in smart stores, such as sensor fusion, smart shelves, and smart cameras. Regarding the different sensor fusion networks, I analyzed how they are applied in different fields, and how they can be built. We used this information later to create our on star topology sensor fusion network for Mercury. In addition to this, I also surveyed different smart cameras and smart shelves proposals, as well as proof of concepts and products that are already operating. This information was particularly useful since some of the smart devices were already operating in smart stores, hence proving its relevance for this case.

After considering the main literature review tasks I carried out, I introduce my most important

contributions regarding the Mercury model and implementation proposal. On the one hand, I was in charged of building the smart shelf using the sensors and the microcontroller contained in the original design. In order to successfully integrate all the components, I had to work not only on the technical decision-taking process but also in the handcrafting of the shelf. Moreover, I had to integrate the shelf with the Mercury Shelves Module. For this purpose, I reviewed the Win32 API to be able to implement serial readouts from the Arduino Uno controller. Besides, I also fully worked on the smart shelf driver based on the HX711 library developed by Bogdan Necula.

On the other hand, I developed the Pose Estimation and Tracking System for the Mercury Computer Vision Module, as well as take the most important decisions with regard to the Camera Manager. In this sense, I had to study different online and offline tracking and pose estimation algorithms, implementing some of them using TensorFlow and deciding which one better fitted our case. I developed also the Node.js back-end server for the REST service of pose estimation. Moreover, I also configured the Bosch Smart Camera FLEXIDOME IP Starlight 8000i, and studied the different protocols to manage it (e.g ONVIF).

Apart from working on the previous modules, I also focused greatly on the Job Dispatcher. I was in charged of designing the computational geometry algorithms related to the linking of human poses with the identified customers, and the estimation of the best candidate of performing an action among all the customers in the store. Moreover, I designed the asynchronous behaviour of the module using jobs to cope with huge workload demands, and therefore making the system more easy to scale up. Additionally, I also implemented these algorithms in Java so that they can be used in our realization of the Mercury smart store.

Another remarkable contribution to the work was the proposal of the retrieval methodology for the CBR implementation in the Explainability Module. I reviewed the scientific literature for this matter, and came up with a feasible implementation for our case, introducing a likelihood and a similarity function which matched our smart store setup. Besides, I also introduced the main ideas concerning the representation of the cases to simplify later the comparison process.

Finally, I analyzed the Mercury system to detect its major flaws. In order to this, I revisited the scenarios that were covered and studied the edge cases for our algorithmic approach. After discovering potential issues and how they could be overcame, I proposed several extensions to the model which could be used to effectively deploy Mercury in production. By considering different methodologies to reproduce 3D views and build fault-tolerant camera networks, I proposed a more resilient smart system based on our original implementation.

B.2 Contributions of Inmaculada Pérez Garbín

Firstly, I was in charge of the front-end side of our implementation of Mercury. To yield this, I dove deep into the scientific literature to come up with the techniques and styling that are used to increase customer engagement. After concluding that a mobile application with the proper layout could effectively fulfill our requirement of enhancing real-time communication with the customer, I studied different possibilities to develop it. Since the resource available for testing this task was an Apple device, I decided to build an iOS application. To achieve that, I had to learned Swift programming language and how to manage Xcode IDE for this purpose, as it has several features and development shortcuts. In addition, to develop the complete application, I introduced the authentication part in the app. I chose the authentication system of Google Firebase, as it provided a robust service and an easy-to-handle API. This made possible to wrap up the application and fully finish it.

Secondly, I developed a Node.js server to enable the communication between the Store Manager

Module with the mobile application. That server allowed the customer to receive the updates in their virtual shopping cart, as well as to request and receive clarifications on their purchase. To accomplish that, I made use of Socket.IO framework, after checking that the integration with Swift of the client side was feasible and simple.

Additionally, I implemented the metadata reading system for the Job Dispatcher Module. That development included making decisions on the metadata format for the Pose Estimation and Tracking Service, and for the Shelf Module. The functions of the system were programmed in Java, incorporating them to the rest of the Job Dispatcher project.

Another interesting contribution to the project was the design of the reuse methodology for the CBR implementation in the Explainability Module. I reviewed different scientific documents regarding CBR and, particularly, templates for the reuse stage of the 4R of the CBR cycle. Finally, I came up with a proper template that includes the required information for the previous and following stages of the cycle.

Regarding the writing process of this document, I analyzed the scientific literature to compile some interesting topics such as recommender systems, biometric technologies for smart stores, and explanation systems for computer vision, in the State of the Art chapter. Moreover, I did a deep research to find the smart stores that are nowadays operating in a production environment. For each store, I investigated the way it work and the technologies that are behind its functionalities, in order to get some clues on what technologies could be performing better for smart stores and, then, studying them in separated sections. After that, I selected a feature that allowed us to classify them into two well-differentiated groups and exposed them in the State of the Art chapter.

Moreover, I designed the use cases of the actions that a customer can perform in our implementation of Mercury. Then, I analyzed our implementation of the Mercury model to trace the scenarios that we were successfully covering, and introduced them in this document. Hence, I was prepared to propose future work for this project and extensions to other domains, that were introduced at the conclusions. In addition, I analyzed whether our objectives were achieved and detail our outputs in the objectives review section.

Finally, another task that I was in charge of was the design of the styling, figures and layout of the document. For this reason, I adjusted the layout of the figures and bibliography, elaborated the templates for the reuse stage of the 4R of the CBR cycle, and created the diagrams that appear in the model and implementation of Mercury.

Bibliography

[1] Daniel Zhang et al. *The AI Index 2021 Annual Report*. 2021. arXiv: 2103.06312 [cs.AI].

[2] M. Dachyar, Teuku Yuri M. Zagloel, and L. Ranjaliba Saragih. "Knowledge growth and development: Internet of Things (IoT) Research, 2006–2018". In: *Heliyon* 5.8 (2019). ISSN: 2405-8440. DOI: https://doi.org/10.1016/j.heliyon.2019.e02264. URL: https://www.sciencedirect.com/science/article/pii/S2405844019359249 (visited on 01/18/2021).

[3] MarketsandMarkets. *Artificial Intelligence Market by Offering (Hardware, Software, Services), Technology (Machine Learning, Natural Language Processing), Deployment Mode, Organization Size, Business Function (Law, Security), Vertical, and Region - Global Forecast to 2026*. 2021.

[4] Fortune Business Insights. *Internet of Things (IoT) Market Size, Share & Covid-19 Impact Analysis, By Component (Platform, Solution and Services), By Platform (Device Management, Cloud Platform, and Network Management), By Solution (Real-Time Streaming Analytics, Security, Data Management, Remote Monitoring), By End-Use (BFSI, Retail, Government, Healthcare, Manufacturing, Agriculture, Sustainable Energy), and Regional Forecast, 2020-2027*. July 2020.

[5] MarketsandMarkets. *Smart Retail Market by System (Smart Payment Systems, Intelligent Vending Machines), Application (Foot-traffic Monitoring, Inventory Management), Retail Offering (Fast-moving Consumer Goods, Hardlines & Leisure Goods), & Geography - Global Forecast to 2025*. 2020.

[6] Bin Guo et al. "DeepStore: Understanding Customer Behaviors in Unmanned Stores". In: *IT Professional* 22.3 (2020), pp. 55–63. DOI: 10.1109/MITP.2019.2928272.

[7] Eleonora Pantano and Charles Dennis. *Smart Retailing: Technologies and Strategies*. 2019. ISBN: 978-3-030-12607-0. DOI: 10.1007/978-3-030-12608-7.

[8] Jianqiang Xu et al. "Design of Smart Unstaffed Retail Shop Based on IoT and Artificial Intelligence". In: *IEEE Access* 8 (2020), pp. 147728–147737. DOI: 10.1109/ACCESS.2020.3014047.

[9] Xiaochen Liu et al. *Grab: Fast and Accurate Sensor Processing for Cashier-Free Shopping*. 2020. arXiv: 2001.01033 [cs.CV].

[10] Carlos Ruiz et al. "Autonomous Inventory Monitoring through Multi-Modal Sensing (AIM3S) for Cashier-Less Stores". In: *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. BuildSys '19. New York, NY, USA: Association for Computing Machinery, 2019, 395–396. ISBN: 9781450370059. DOI: 10.1145/3360322.3361018. URL: https://doi.org/10.1145/3360322.3361018 (visited on 04/30/2021).

[11] You-Chiun Wang and Chang-Chen Yang. "3S-cart: A Lightweight, Interactive Sensor-Based Cart for Smart Shopping in Supermarkets". In: *IEEE Sensors Journal* 16.17 (2016), pp. 6774–6781. DOI: 10.1109/JSEN.2016.2586101.

[12] Marta Sabou et al. "Position paper on realizing smart products: Challenges for semantic web technologies". In: *Proceedings of A workshop of the 8th International Semantic Web Conference, ISWC 2009*. 2009, pp. 135–147.

[13] C. Gutiérrez et al. "Providing a Consensus Definition for the Term "Smart Product"". In: *2013 20th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*. 2013, pp. 203–211. DOI: 10.1109/ECBS.2013.26.

[14] Max Mühlhäuser. "Smart Products: An Introduction". In: *Constructing Ambient Intelligence*. 2008, pp. 158–164. DOI: 10.1007/978-3-540-85379-4_20.

[15] Eleonora Pantano and Harry Timmermans. "What is Smart for Retailing?" In: *Procedia Environmental Sciences* 22 (2014), pp. 101–107. ISSN: 1878-0296. DOI: https://doi.org/10.1016/j.proenv.2014.11.010. URL: https://www.sciencedirect.com/science/article/pii/S1878029614001571.

[16] Sujana Adapa et al. "Examining the antecedents and consequences of perceived shopping value through smart retail technology". In: *Journal of Retailing and Consumer Services* 52 (2020), p. 101901. ISSN: 0969-6989. DOI: https://doi.org/10.1016/j.jretconser.2019.101901. URL: https://www.sciencedirect.com/science/article/pii/S0969698919302978 (visited on 03/24/2021).

[17] Victoria Shannon. *Do-it-yourself checkout is spreading in Europe*. 2020. URL: {https://www.nytimes.com/2004/09/20/business/worldbusiness/wireless-doityourself-checkout-is-spreading-in-europe.html} (visited on 01/22/2021).

[18] Amazon. *Amazon Go*. 2018. URL: https://www.amazon.com/-/es/b?ie=UTF8&node=16008589011 (visited on 01/21/2021).

[19] NTT Data. *Announcing the introduction of Facial Authentication and Dynamic Pricing at Catch&Go™ digital grocery stores*. 2020. URL: https://www.nttdata.com/global/en/media/press-release/2020/february/announcing-the-introduction-of-facial-authentication-and-dynamic-pricing (visited on 01/21/2021).

[20] The Business Research Company. *Retail Global Market Report 2021: COVID-19 Impact and Recovery to 2030*. 2021.

[21] Praveen Adhi et al. *A transformation in store*. 2019.

[22] 451 Research. *Retail Reimagined Experience – Not Price – Is the Battleground of the Future*. Tech. rep. Adyen, 2018.

[23] Amazon. *Just Walk-Out Technology*. 2020. URL: {https://justwalkout.com/} (visited on 01/21/2021).

[24] Werner Reinartz, Nico Wiegand, and Monika Imschloss. "The impact of digital transformation on the retailing value chain". In: *International Journal of Research in Marketing* 36.3 (2019). Marketing Perspectives on Digital Business Models, pp. 350–366. ISSN: 0167-8116. DOI: https://doi.org/10.1016/j.ijresmar.2018.12.002. URL: https://www.sciencedirect.com/science/article/pii/S0167811618300739 (visited on 02/10/2021).

[25] Jaime Toplin. *Amazon is opening new locations of its 4-star and Books stores*. 2019. URL: https://www.businessinsider.com/amazon-expands-physical-retail-footprint-2019-8 (visited on 01/21/2021).

[26] Ahmed Nabil Belbachir. *Smart cameras*. Springer, 2010.

[27] Fábio Dias Real and François Berry. "Smart Cameras: Technologies and Applications". In: Springer US, 2009, pp. 35–50. DOI: 10.1007/978-1-4419-0953-4_3. URL: https://doi.org/10.1007/978-1-4419-0953-4_3 (visited on 02/09/2021).

[28] Reem Alshalawi and Mohamed Osama Khozium. "A Case Study of IP Camera Monitoring Traffic Verification". In: *International Journals of Advanced Research in Computer Science and Software Engineering* 8.12 (2018). DOI: https://doi.org/10.23956/ijarcsse.v8i12.924. URL: http://www.ijarcsse.com/index.php/ijarcsse/article/view/924/0 (visited on 02/09/2021).

[29] Reem Alshalawi and Turki Alghamdi. "Forensic tool for wireless surveillance camera". In: 2017, pp. 536–540. DOI: 10.23919/ICACT.2017.7890148.

[30] Gradimirka Popovic et al. "Overview , Characteristics and Advantages of IP Camera Video Surveillance Systems Compared to Systems with other Kinds of Camera". In: 2013.

[31] Bosch Security Systems. *Technical Note. FW 6.30 Intelligent Tracking*. Tech. rep. Robert-Bosch-Ring 5, Grasbrunn. Germany: Bosch Security Systems, 2016. URL: https://resources-boschsecurity-cdn.azureedge.net/public/documents/TN_VCA_intelligent_t_WhitePaper_enUS_23419022347.pdf (visited on 02/09/2021).

[32] Joonsoo Lee and Al Bovik. "Chapter 19 - Video Surveillance". In: *The Essential Guide to Video Processing*. Ed. by Al Bovik. Boston: Academic Press, 2009, pp. 619–651. ISBN: 978-0-12-374456-2. DOI: https://doi.org/10.1016/B978-0-12-374456-2.00022-0. URL: https://www.sciencedirect.com/science/article/pii/B9780123744562000220 (visited on 02/09/2021).

[33] Jebah Jaykumar and Abishlin Blessy. "Secure Smart Environment Using IOT based on RFID". In: *International Journal of Computer Science and Information Technologies* 5.2 (2014), pp. 2493–2496. ISSN: 0975-9646. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.590.9212&rep=rep1&type=pdf (visited on 02/10/2021).

[34] Kamran Ahsan, Hanifa Shah, and Paul Kingston. *RFID Applications: An Introductory and Exploratory Study*. 2010.

[35] Chris West et al. *Walmart: RFID Technolgy*. 2013. URL: https://walmartsupplychain.weebly.com/rfid-technology.html (visited on 02/10/2021).

[36] IBM. *IBM RFID Commercial-The Future Supermarket*. YouTube. 2013. URL: https://www.youtube.com/watch?v=wzFhBGKU6HA (visited on 02/10/2021).

[37] Gianna Lise Puerini, Dilup Kumar, and Steven Kessel. *Transitioning items from a materials handling facility*. 2015. URL: https://patents.google.com/patent/US20150012396A1/en (visited on 02/13/2021).

[38] Lauren Moore. *Walmart and RFID: The Relationship That put RFID on the Map*. 2019. URL: https://www.atlasrfidstore.com/rfid-insider/walmart-and-rfid-the-relationship-that-put-rfid-on-the-map (visited on 02/10/2021).

[39] Amin Rida et al. "Paper-Based Ultra-Low-Cost Integrated RFID Tags for Sensing and Tracking Applications". In: *2007 Proceedings 57th Electronic Components and Technology Conference*. 2007, pp. 1977–1980. DOI: 10.1109/ECTC.2007.374072.

[40] Moutaz Haddara and Anna Staaby. "RFID Applications and Adoptions in Healthcare: A Review on Patient Safety". In: *Procedia Computer Science* 138 (2018), pp. 80–88. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.10.012. URL: https://www.sciencedirect.com/science/article/pii/S1877050918316430 (visited on 02/10/2021).

[41] A.W.-K. Kong and D. Zhang. "Competitive coding scheme for palmprint verification". In: *Proceedings of the 17th International Conference on Pattern Recognition.* Vol. 1. 2004, pp. 520–523. DOI: 10.1109/ICPR.2004.1334184.

[42] Adams Kong, David Zhang, and Mohamed Kamel. "Palmprint identification using feature-level fusion". In: *Pattern Recognition* 39.3 (2006), pp. 478–487. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2005.08.014. URL: https://www.sciencedirect.com/science/article/pii/S0031320305003766 (visited on 05/15/2021).

[43] Peter Jones et al. "Biometrics in retailing". In: *International Journal of Retail & Distribution Management* 35.3 (2007), pp. 217–222. DOI: 10.1108/09590550710735077. URL: https://doi.org/10.1108/09590550710735077 (visited on 05/15/2021).

[44] Amazon. *Amazon One*. 2020. URL: https://one.amazon.com/ (visited on 05/15/2021).

[45] Rakesh Satapathy, Srikanth Prahlad, and Vijay Kaulgud. "Smart Shelfie – Internet of Shelves: For Higher On-Shelf Availability". In: *2015 IEEE Region 10 Symposium.* 2015, pp. 70–73. DOI: 10.1109/TENSYMP.2015.9.

[46] Michele Merler, Carolina Galleguillos, and Serge Belongie. "Recognizing Groceries in situ Using in vitro Training Data". In: *2007 IEEE Conference on Computer Vision and Pattern Recognition.* 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383486.

[47] Alessio Tonioni, Eugenio Serra, and Luigi Di Stefano. "A deep learning pipeline for product recognition on store shelves". In: *2018 IEEE International Conference on Image Processing, Applications and Systems (IPAS).* 2018, pp. 25–31. DOI: 10.1109/IPAS.2018.8708890.

[48] Deepti Gupta et al. "Future Smart Connected Communities to Fight COVID-19 Outbreak". In: *Internet of Things* 13 (2021), p. 100342. ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2020.100342. URL: https://www.sciencedirect.com/science/article/pii/S2542660520301736 (visited on 04/04/2021).

[49] W. Elmenreich and S. Pitzek. "Using sensor fusion in a time-triggered network". In: *IECON'01. 27th Annual Conference of the IEEE Industrial Electronics Society (Cat. No.37243).* Vol. 1. 2001, 369–374 vol.1. DOI: 10.1109/IECON.2001.976510.

[50] H. F. Durrant-Whyte. "Sensor Models and Multisensor Integration". In: *Int. J. Rob. Res.* 7.6 (1988), 97–113. ISSN: 0278-3649. DOI: 10.1177/027836498800700608. URL: https://doi.org/10.1177/027836498800700608 (visited on 04/02/2021).

[51] Diego Galar and Uday Kumar. "Chapter 1 - Sensors and Data Acquisition". In: *eMaintenance.* Ed. by Diego Galar and Uday Kumar. Academic Press, 2017, pp. 1–72. ISBN: 978-0-12-811153-6. DOI: https://doi.org/10.1016/B978-0-12-811153-6.00001-4. URL: https://www.sciencedirect.com/science/article/pii/B9780128111536000014.

[52] A. Hoover and B.D. Olsen. "Sensor network perception for mobile robotics". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065).* Vol. 1. 2000, pp. 342–347. DOI: 10.1109/ROBOT.2000.844080.

[53] A. Visser and F. C. A. Groen. *Organisation and Design of Autonomous Systems.* 1999.

[54] Francesco Ricci, Lior Rokach, and Bracha Shapira. "Introduction to Recommender Systems Handbook". In: *Recommender Systems Handbook.* Springer US, 2010, pp. 1–35. DOI: 10.1007/978-0-387-85820-3_1. URL: https://doi.org/10.1007/978-0-387-85820-3_1.

[55] Linyuan Lü et al. "Recommender systems". In: *Physics Reports* 519.1 (2012), pp. 1–49. ISSN: 0370-1573. DOI: https://doi.org/10.1016/j.physrep.2012.02.006. URL: https://www.sciencedirect.com/science/article/pii/S0370157312000828.

[56] Gediminas Adomavicius and Alexander Tuzhilin. "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions". In: *IEEE Transactions on Knowledge and Data Engineering* 17.6 (2005), pp. 734–749. DOI: 10.1109/TKDE.2005.99.

[57] Daniel Zeng et al. "Location-Aware Real-Time Recommender Systems for Brick-and-Mortar Retailers". In: *INFORMS Journal on Computing* (2021). DOI: 10.1287/ijoc.2020.1020. URL: https://doi.org/10.1287/ijoc.2020.1020.

[58] Caper AI. *Caper AI Cart*. 2020. URL: https://www.caper.ai/cart (visited on 05/31/2021).

[59] Yvette Tan. *China's version of Amazon's cashier-less store is here*. 2017. URL: https://mashable.com/2017/07/12/taobao-china-store/?europe=true#57OXUGTQVaqg (visited on 01/21/2021).

[60] V Chvátal. "A combinatorial theorem in plane geometry". In: *Journal of Combinatorial Theory, Series B* 18.1 (1975), pp. 39–41. ISSN: 0095-8956. DOI: https://doi.org/10.1016/0095-8956(75)90061-1. URL: https://www.sciencedirect.com/science/article/pii/0095895675900611 (visited on 04/24/2021).

[61] N. Chesnokov. "The Art Gallery Problem: An Overview and Extension to Chromatic Coloring and Mobile Guards". In: 2018.

[62] Andreas Bärtschi. "Coloring Variations of the Art Gallery Problem". In: 2011.

[63] M. Palsson and J. Stahl. "The Camera Placement Problem - An art gallery problem variation". In: 2008.

[64] Jinwoo Kim et al. "Systematic Camera Placement Framework for Operation-Level Visual Monitoring on Construction Jobsites". In: *Journal of Construction Engineering and Management* 145.4 (2019), pp. 0401–9019. DOI: 10.1061/(ASCE)CO.1943-7862.0001636. eprint: https://ascelibrary.org/doi/pdf/10.1061/\%28ASCE\%29CO.1943-7862.0001636. URL: https://ascelibrary.org/doi/abs/10.1061/\%28ASCE\%29CO.1943-7862.0001636 (visited on 04/24/2021).

[65] Rigoberto Juarez-Salazar, Juan Zheng, and Victor H. Diaz-Ramirez. "Distorted pinhole camera modeling and calibration". In: *Appl. Opt.* 59.36 (2020), pp. 11310–11318. DOI: 10.1364/AO.412159. URL: http://ao.osa.org/abstract.cfm?URI=ao-59-36-11310 (visited on 04/24/2021).

[66] Dimo Chotrov et al. "Mixed-Reality Spatial Configuration with a ZED Mini Stereoscopic Camera". In: 2018.

[67] W. N. Martin. *Motion understanding : robot and human vision*. Boston: Kluwer Academic Publishers, 1988. ISBN: 978-0-89838-258-7.

[68] Xiaolong Wang. "Learning and Reasoning with Visual Correspondence in Time". PhD thesis. Pittsburgh, PA: Carnegie Mellon University, 2019.

[69] Henning Zimmer. "Correspondence problems in computer vision". PhD thesis. Jan. 2012.

[70] Olivier Faugeras, Quang-Tuan Luong, and T. Papadopoulou. *The Geometry of Multiple Images: The Laws That Govern The Formation of Images of A Scene and Some of Their Applications*. Cambridge, MA, USA: MIT Press, 2001. ISBN: 0262062208.

[71] B. D. Lucas and T. Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *IJCAI*. 1981.

[72] James Black and Tim Ellis. "Multi-camera image measurement and correspondence". In: *Measurement* 32 (July 2002), pp. 61–71. DOI: 10.1016/S0263-2241(01)00050-1.

[73] Xiaogang Wang, Kinh Tieu, and W. Eric L. Grimson. "Correspondence-free multi-camera activity analysis and scene modeling". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587722.

[74] V.I. Morariu and Octavia Camps. "Modeling Correspondences for Multi-Camera Tracking Using Nonlinear Manifold Learning and Target Dynamics". In: vol. 1. July 2006, pp. 545–552. ISBN: 0-7695-2597-0. DOI: 10.1109/CVPR.2006.189.

[75] S. Roy and I.J. Cox. "A maximum-flow formulation of the N-camera stereo correspondence problem". In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, pp. 492–499. DOI: 10.1109/ICCV.1998.710763.

[76] Zhengyou Zhang. "Determining the Epipolar Geometry and its Uncertainty: A Review". In: *International Journal of Computer Vision* 27.2 (1998), pp. 161–195. DOI: 10.1023/a:1007941100561. URL: https://doi.org/10.1023/a:1007941100561 (visited on 04/24/2021).

[77] Longuet-Higgins, H. C. "A computer algorithm for reconstructing a scene from two projections". In: *Nature* 293.5828 (1981), pp. 133–135. DOI: 10.1038/293133a0. URL: https://doi.org/10.1038/293133a0 (visited on 04/24/2021).

[78] Richard I. Hartley. "In Defense of the Eight-Point Algorithm". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 19.6 (1997), 580–593. ISSN: 0162-8828. DOI: 10.1109/34.601246. URL: https://doi.org/10.1109/34.601246 (visited on 05/10/2021).

[79] JK Aggarwal and N Nandhakumar. "On the computation of motion from sequences of images-a review". In: *Proceedings of the IEEE* 76.8 (1988), pp. 917–935.

[80] Thomas S Huang and Arun N Netravali. "Motion and structure from feature correspondences: A review". In: *Advances In Image Processing And Understanding: A Festschrift for Thomas S Huang* (2002), pp. 331–347.

[81] John Hershberger and Subhash Suri. "An Optimal Algorithm for Euclidean Shortest Paths in the Plane". In: *SIAM Journal on Computing* 28.6 (1999), pp. 2215–2256. DOI: 10.1137/s0097539795289604. URL: https://doi.org/10.1137/s0097539795289604 (visited on 05/10/2021).

[82] Gregory Shakhnarovich, Trevor Darrell, and P. Indyk. "Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)". In: 2005.

[83] Kaziwa Saleh, Sandor Szenasi, and Zoltan Vamossy. "Occlusion Handling in Generic Object Detection: A Review". In: *2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI)* (2021). DOI: 10.1109/sami50585.2021.9378657. URL: http://dx.doi.org/10.1109/SAMI50585.2021.9378657 (visited on 05/11/2021).

[84] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[85] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014.

[86] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014.

[87] Jianwei Yang et al. "Embodied Amodal Recognition: Learning to Move to Perceive Objects". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 2040–2050. DOI: 10.1109/ICCV.2019.00213.

[88] Patrick Follmann et al. *Learning to See the Invisible: End-to-End Trainable Amodal Instance Segmentation*. 2018.

[89] Yonglong Tian et al. "Deep Learning Strong Parts for Pedestrian Detection". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1904–1912. DOI: 10.1109/ICCV.2015.221.

[90] Junhyug Noh et al. "Improving Occlusion and Hard Negative Handling for Single-Stage Pedestrian Detectors". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 966–974. DOI: 10.1109/CVPR.2018.00107.

[91] Yongjun Li et al. "YOLO-ACN: Focusing on small target and occluded object detection". In: *IEEE* (2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.3046515.

[92] Guoxu Liu et al. "YOLO-Tomato: A Robust Algorithm for Tomato Detection Based on YOLOv3". In: *Sensors* 20.7 (2020), p. 2145. DOI: 10.3390/s20072145. URL: https://doi.org/10.3390/s20072145 (visited on 04/26/2021).

[93] Yan Zhu et al. "Semantic amodal segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1464–1472.

[94] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV].

[95] Ross Girshick. *Fast R-CNN*. 2015.

[96] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016.

[97] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *Lecture Notes in Computer Science* (2016), 21–37. ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2 (visited on 05/01/2021).

[98] Shifeng Zhang et al. *Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection*. 2020.

[99] Songtao Liu, Di Huang, and Yunhong Wang. *Learning Spatial Fusion for Single-Shot Object Detection*. 2019.

[100] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016.

[101] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: *CoRR* abs/2004.10934 (2020). URL: https://arxiv.org/abs/2004.10934 (visited on 04/06/2021).

[102] Wafae Mrabti et al. "Human motion tracking: A comparative study". In: *Procedia Computer Science* 148 (2019). The second international conference on intelligent computing in data science (ICDS2018), pp. 145–153. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.01.018. URL: https://www.sciencedirect.com/science/article/pii/S1877050919300183 (visited on 04/03/2021).

[103] Wenhan Luo et al. "Multiple object tracking: A literature review". In: *Artificial Intelligence* 293 (2021), p. 103448. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2020.103448. URL: https://www.sciencedirect.com/science/article/pii/S0004370220301958 (visited on 05/02/2021).

[104] Kaihua Zhang, Lei Zhang, and Ming-Hsuan Yang. "Real-Time Compressive Tracking". In: *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 864–877.

[105] Digital Commerce 360. *Ahold Delhaize tests a cashier-free food store concept*. 2019. URL: https://www.digitalcommerce360.com/2019/11/26/ahold-delhaize-tests-a-cashier-free-food-store-concept/ (visited on 01/21/2021).

[106] Nicolai Wojke and Alex Bewley. "Deep Cosine Metric Learning for Person Re-identification". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 748–756. DOI: 10.1109/WACV.2018.00087.

[107] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple Online and Realtime Tracking with a Deep Association Metric". In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962.

[108] Alex Bewley et al. "Simple online and realtime tracking". In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

[109] Wanli Ouyang, Xiao Chu, and Xiaogang Wang. "Multi-source Deep Learning for Human Pose Estimation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 2337–2344. DOI: 10.1109/CVPR.2014.299.

[110] Xavier Alameda-Pineda et al. "Chapter 14 - SALSA: A Multimodal Dataset for the Automated Analysis of Free-Standing Social Interactions". In: *Group and Crowd Behavior for Computer Vision*. Ed. by Vittorio Murino et al. Academic Press, 2017, pp. 321–340. ISBN: 978-0-12-809276-7. DOI: https://doi.org/10.1016/B978-0-12-809276-7.00017-5. URL: https://www.sciencedirect.com/science/article/pii/B9780128092767000175 (visited on 04/02/2021).

[111] Standard AI. *Standard Cognition - AI-powered checkout*. 2017. URL: https://standard.ai/ (visited on 01/22/2021).

[112] Dario Pavllo et al. "3D Human Pose Estimation in Video With Temporal Convolutions and Semi-Supervised Training". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 7745–7754.

[113] Google. *TensorFlow, Mobile and IoT. Pose Estimation*. Google, 2021. URL: https://www.tensorflow.org/lite/examples/pose_estimation/overview (visited on 04/02/2021).

[114] George Papandreou et al. "PersonLab: Person Pose Estimation and Instance Segmentation with a Bottom-Up, Part-Based, Geometric Embedding Model". In: *CoRR* abs/1803.08225 (2018). URL: http://arxiv.org/abs/1803.08225 (visited on 04/02/2021).

[115] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[116] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[117] Wojciech Samek and Klaus-Robert Müller. "Towards Explainable Artificial Intelligence". In: 2019, pp. 5–22. ISBN: 978-3-030-28953-9. DOI: 10.1007/978-3-030-28954-6_1.

[118] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. https://christophm.github.io/interpretable-ml-book/. 2019.

[119] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. "Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models". In: *ITU Journal: ICT Discoveries - Special Issue 1 - The Impact of Artificial Intelligence (AI) on Communication Networks and Services* 1 (2017), pp. 1–10.

[120] Sen Yang et al. "TransPose: Towards Explainable Human Pose Estimation by Transformer". In: *ArXiv* abs/2012.14214 (2020).

[121] Neil Robertson, Ian Reid, and Michael Brady. "Automatic Human Behaviour Recognition and Explanation for CCTV Video Surveillance". In: *Security Journal* 21 (2008). DOI: 10.1057/palgrave.sj.8350053.

[122] Marco Rigolli and Michael Brady. "Towards a behavioural traffic monitoring system". In: 2005, pp. 449–454. DOI: 10.1145/1082473.1082542.

[123] Brian Heater. *Amazon launches a beta of Go, a cashier-free, app-based food shopping experience*. 2016. URL: https://techcrunch.com/2016/12/05/amazon-go/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAJGlkutr-XERVEo2maMD7GjSqci7JyvAkYIn6L2GJerRLWgus8VW93rWb0lthBDZLaneymTGgy1RL_UAZRLmAHa2oyoPpU-g9Q6d7Cegt7keGkAWXObAG_zrYnD2QwdYt0D6f0EzkIfEwlii5cr7tS4VyJg4-LrmvmnG9SUyrYcB (visited on 01/21/2021).

[124] Conner Forrest. *Amazon Go launches: The automated retail revolution begins*. 2018. URL: https://www.techrepublic.com/article/amazon-go-launches-the-automated-retail-revolution-begins/ (visited on 01/21/2021).

[125] Jeff Wells. *Amazon plans more Go Grocery stores*. 2020. URL: https://www.grocerydive.com/news/amazon-plans-more-go-grocery-stores/581046/ (visited on 01/21/2021).

[126] Amazon. *Amazon Go Grocery*. 2019. URL: https://www.amazon.com/-/es/b?ie=UTF8&node=20931388011 (visited on 01/21/2021).

[127] K. Wankhede, B. Wukkadada, and V. Nadar. "Just Walk-Out Technology and its Challenges: A Case of Amazon Go". In: *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. 2018, pp. 254–257. DOI: 10.1109/ICIRCA.2018.8597403.

[128] Amazon. *Amazon Dash Cart*. 2020. URL: https://www.amazon.com/b?ie=UTF8&node=21289116011 (visited on 01/21/2021).

[129] Amazon. *Amazon Fresh Locations*. 2020. URL: https://www.amazon.com/-/es/b/ref=s9_acss_bw_cg_ABFYSA_2c1_w?node=17608448011&pf_rd_m=ATVPDKIKX0DER&pf_rd_s=merchandised-search-1&pf_rd_r=8MM1K0F5HPNCTFKFE169&pf_rd_t=101&pf_rd_p=b710ddf5-644b-4f70-98bd-359bab72a28d&pf_rd_i=17608448011#AmazonFreshLocations (visited on 01/21/2021).

[130] Bridget Goldschmidt. *Ahold Delhaize USA Pilots Frictionless Store*. 2019. URL: https://progressivegrocer.com/ahold-delhaize-usa-pilots-frictionless-store (visited on 01/21/2021).

[131] Krishna Thakker. *A look inside Ahold Delhaize's 24/7 frictionless store*. 2020. URL: https://www.grocerydive.com/news/a-look-inside-ahold-delhaizes-247-frictionless-store/570385/ (visited on 01/21/2021).

[132] elEconomista. *Llega el primer supermercado sin cajas (y sin colas) de Europa: así es el nuevo Continente Labs*. 2021. URL: https://www.eleconomista.es/empresas-finanzas/noticias/11244811/05/21/Llega-el-primer-supermercado-sin-cajas-y-sin-colas-de-Europa-asi-es-el-nuevo-Continente-Labs.html (visited on 05/31/2021).

[133] Continente. *Continente Labs*. 2021. URL: https://www.youtube.com/watch?v=0lE8b1KR_10 (visited on 05/31/2021).

[134] Meios & Publicidade. *230 câmaras, 400 sensores e zero caixas de pagamento. Como funciona a primeira loja Continente Labs*. 2021. URL: https://www.meiosepubli

cidade.pt/2021/05/230-camaras-400-sensores-e-zero-caixas-de-pagamento-como-funciona-a-primeira-loja-continente-labs/ (visited on 05/31/2021).

[135] Alicia Davara et al. *Decathlon implanta tecnología RFID en las cajas de autoservicio*. 2019. URL: https://www.distribucionactualidad.com/asi-funciona-tecnologia-rfid-decathlon/ (visited on 01/21/2021).

[136] Adrian Beck. "Measuring the Impact of RFID in Retailing: Keys Lessons from 10 Case-study Companies". In: 2018. DOI: 10.13140/RG.2.2.27522.32968.

[137] Albert Heijn. *Everything about Albert Heijn*. 2020. URL: https://www.ah.nl/over-ah (visited on 01/21/2021).

[138] Werken bij NS. *Working at AH to go*. Werken bij NS, 2021. URL: https://werkenbijns.nl/formules/ah-to-go/ (visited on 01/21/2021).

[139] NU.nl. *Albert Heijn stops with AH to go in Germany*. NU.nl, 2017. URL: https://www.nu.nl/economie/5047925/albert-heijn-stopt-met-ah-to-go-in-duitsland.html (visited on 01/21/2021).

[140] Tang Shihua. *Staffless Convenience Chain 17 Shandian Gets A-Round Funding*. 2018. URL: https://www.yicaiglobal.com/news/staffless-convenience-chain-17-shandian-gets-a-round-funding (visited on 01/22/2021).

[141] Inokyo. *Inokyo - How It Works*. 2018. URL: https://www.inokyo.com/ (visited on 01/22/2021).

[142] Kaila Imada. *FamilyMart opens a self-checkout convenience store with no cashier*. 2021. URL: https://www.timeout.com/tokyo/news/familymart-opens-a-self-checkout-convenience-store-with-no-cashier-040621 (visited on 01/22/2021).

[143] Josh Constine. *Meet Caper, the AI self-checkout shopping cart*. 2019. URL: https://techcrunch.com/2019/01/10/caper-shopping-cart/ (visited on 01/22/2021).

[144] Caper AI. *Caper AI Counter*. 2020. URL: https://www.caper.ai/counter (visited on 01/22/2021).

[145] W. Lu and Y. Tan. "A color histogram based people tracking system". In: *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196)* 2 (2001), pp. 137–140.

[146] Z. Cao et al. "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[147] Ian Watson. "Case-Based Reasoning is a Methodology not a Technology". In: *Research and Development in Expert Systems XV*. Ed. by Roger Miles, Michael Moulton, and Max Bramer. London: Springer London, 1999, pp. 213–223. ISBN: 978-1-4471-0835-1.

[148] Neel Joshi et al. "Synthetic Aperture Tracking: Tracking through Occlusions". In: *2007 IEEE 11th International Conference on Computer Vision*. 2007, pp. 1–8. DOI: 10.1109/ICCV.2007.4409032.

[149] GitHub. *GitHub: Where the world builds software*. 2008. URL: https://github.com/ (visited on 10/27/2020).

[150] Ivan Muller et al. "Load cells in force sensing analysis – theory and a novel application". In: *IEEE Instrumentation Measurement Magazine* 13.1 (2010), pp. 15–19. DOI: 10.1109/MIM.2010.5399212.

[151] Bogdan Necula. *An Arduino library to interface the Avia Semiconductor HX711 24-Bit Analog-to-Digital Converter (ADC) for Weight Scales*. https://github.com/bogde/HX711. 2018.

[152] Arduino. *Arduino Serial Reference*. 2019. URL: https://www.arduino.cc/reference/en/language/functions/communication/serial/ (visited on 02/15/2021).

[153] Dawoud Dawoud. *Serial communication protocols and standards : RS232/485, UART/USART, SPI, USB, INSTEON, Wi-Fi and WiMAX*. Aalborg: River Publishers, 2020. ISBN: 8770221545.

[154] Microsoft. *Programming reference for the Win32 API - Win32 apps*. 2019. URL: https://docs.microsoft.com/en-us/windows/win32/api/ (visited on 02/15/2021).

[155] Elena Sőnmez, Ceren Melek, and Songul Varlı. "A Survey of Product Recognition in Shelf Images". In: 2017. DOI: 10.1109/UBMK.2017.8093584.

[156] Roy Thomas Fielding and Richard N. Taylor. "Architectural Styles and the Design of Network-Based Software Architectures". PhD thesis. 2000. ISBN: 0599871180.

[157] ONVIF. *ONVIF website*. 2020. URL: https://www.onvif.org/ (visited on 11/30/2020).

[158] Harald van Heerde, Isaac Dinner, and Scott Neslin. "Creating Customer Engagement Via Mobile Apps: How App Usage Drives Purchase Behavior". In: *Working paper* (2015). DOI: 10.2139/ssrn.2669817.

[159] Laura Silver. *Smartphone Ownership Is Growing Rapidly Around the World, but Not Always Equally*. 2019. URL: https://www.pewresearch.org/global/2019/02/05/smartphone-ownership-is-growing-rapidly-around-the-world-but-not-always-equally/ (visited on 02/27/2021).

[160] Google. *Firebase Use Cases*. 2014. URL: https://firebase.google.com/use-cases?hl=es-419 (visited on 02/27/2021).

[161] Agnar Aamodt and Enric Plaza. "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches". In: *AI Commun.* 7.1 (1994), 39–59. ISSN: 0921-7126.

[162] Ali Montazemi and Kalyan Gupta. "A framework for retrieval in case-based reasoning systems". In: *Annals of Operations Research* 72 (Jan. 1997), pp. 51–73. DOI: 10.1023/A:1018960607821.

[163] Jennifer Bray, Charles F. Sturman, and Tom Siep. *Bluetooth: Connect Without Cables*. Prentice Hall, 2002.

[164] J. Hallberg, M. Nilsson, and K. Synnes. "Positioning with Bluetooth". In: *10th International Conference on Telecommunications, 2003. ICT 2003.* Vol. 2. 2003, pp. 954–958. DOI: 10.1109/ICTEL.2003.1191568.

[165] Samuel Opoku. "An Indoor Tracking System Based on Bluetooth Technology". In: (2012).

[166] Mandeep Kaur et al. "RFID Technology Principles, Advantages, Limitations & Its Applications". In: *International Journal of Computer and Electrical Engineering* (2011), pp. 151–157.

[167] Asela Gunawardana and Guy Shani. "A Survey of Accuracy Evaluation Metrics of Recommendation Tasks". In: *J. Mach. Learn. Res.* 10 (Dec. 2009), 2935–2962. ISSN: 1532-4435.

[168] Albrecht Beutelspacher. *Projective geometry : from foundations to applications*. Cambridge New York, NY, USA: Cambridge University Press, 1998. ISBN: 978-0-521-48364-3.

[169] R.A. Jarvis. "On the identification of the convex hull of a finite set of points in the plane". In: *Information Processing Letters* 2.1 (1973), pp. 18–21. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(73)90020-3. URL: https://www.sciencedirect.com/science/article/pii/0020019073900203 (visited on 05/28/2021).